ADVANCED TECHNIQUES FOR OIL RESERVOIR SIMULATION: DISCRETE FRACTURE MODEL AND PARALLEL IMPLEMENTATION

by

Jong Gyun Kim

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Chemical and Fuels Engineering

The University of Utah

December 1999

TABLE OF CONTENTS

ABSTRACT

Major reservoir development decisions are often based on results of numerical simulation of oil reservoirs. Partial differential equations (PDEs) derived from material balance coupled with Darcy's law are solved in the simulator. The speed and efficiency of the simulator are determined to a large extend by the efficiency of the linear solver, which is usually ultimate step in the solution of the system PDEs. In this work, the efficiency of the following linear solvers was examined on shared and distributed memory parallel machines: the line successive over relaxation, Bi-conjugate gradient stabilized (Bi-CGSTAB), the restarted generalized minimum residual and the transpose free quasi minimal residual (TFQMR) methods. All the methods except the Bi-CGSTAB were stable and exhibited fast convergence behavior. Of the conjugate gradient like methods, TFQMR appeared to be the best. Considerable speedup (factor of about 4 with eight processors) was observed on shared and distributed memory machines.

Naturally fractured oil reservoirs have been simulated using the conventional dual porosity (dual permeability) and single porosity models. Even though these techniques have served the practical purpose of simulating real reservoirs, it is recognized that it is impossible to represent fractures, spatially accurately through these approaches. A discrete fracture model was developed as an alternative to these conventional simulation methods. In a two-dimensional test problem, lines represented fractures while the matrix space was discretized into triangular elements. Results from this model were compared

to simulation results using a variety of homogenization methods. At high permeability contrast (between the matrix and the fracture) and high flow rates (injection), the discrete fracture model captured the explicit features of the fractures, which the homogenization method failed to reproduce. Fully implicit formulation of the discrete fracture model resulted in a system of nonlinear equations, which were solved using the inexact Newton's scheme. The inexact scheme, which utilized the agreement between the function and its local linear model, was found to be significantly better than the methods that arbitrarily fixed the level of inexactness. A successful parallel implementation of the discrete fracture model using domain decomposition was achieved on both SGI Power Challenge and SGI Origin 2000.

LIST OF TABLES

LIST OF FIGURES

LIST OF SYSMBOLS

Notation

$(Ac)_i$ =Cross sectional area of grid block *i*
B_l = Fluid formation volume factor of phase *l*
$k_{absolute}$ = Absolute permeability
k_{rl} = Relative permeability of phase *l*
P_c = Capillary pressure
P_l = Pressure of phase *l*
q_l / ρ_{lSC} = Volume produced or injected (of phase l) per unit time per unit reservoir volume
R_{so} = Solution gas oil ratio
S_l = Saturation of phase *l*
t = Time
T_l = Transmissibility of phase *l*
x^* = Dimension in local coordinate system for fracture elements
x,y = Dimensions in the Cartesian coordinate system
Z = Elevation
γ_l = Density of phase *l* in terms of pressure/distance
μ_l = Viscosity of phase *l*
ϕ = Porosity
Φ = Shape function for finite element
Ω = Finite element domain
∇ = Difference operator in Cartesian coordinate system
∇^* = Difference operator in the local (fracture) coordinate system

Subscript

o = Oil phase
w = Related to water phase
f = Related to fracture
RC = Reservoir condition
STC = Stock tank or standard condition

ACKNOWLEDGEMENTS

First of all, my thanks go to Dr. Milind D. Deo for being a constant source of encouragement in my graduate study. Most importantly, he has been my mentor for the last three years, providing important corrections, ideas, and suggestions to the final release of this dissertation. I would also like to thank the committee professors, Dr. Dennis L. Nielson, Dr. Francis V. Hanson, Dr. Raj K. Rajamani, and Dr. Junior D. Seader for their valuable supervision. I also wish to thank Dr. Craig B. Forster of the Department of Geology and Geophysics, University of Utah and Dr. Joseph B. Koebbe of the Department of Mathematics, Utah Sate University for their input of simulation data. I would never forget the invaluable time, which my colleagues (Hongmei Huang and Rajesh Pawar) did provide in the multicultural environment of our lab. I hope the best luck to Mr. Kyeong Seok Oh, who just started his research program in our lab. Most of all, I am thankful to my wife, Kyoungaeh Baek and especially my one-year-old daughter, Hali S. Kim volunteering to test the creations of her father and mother. Without their love, patience, and support, the successful accomplishment of this work would have been impossible. Finally, I would like to thank GOD for lifting my spirit and sharing the happiness of my family.

CHAPTER 1

INTRODUCTION

Oil reservoir simulation is an established technique used in the petroleum industry to assess the quality of reservoirs and to plan production strategies. Classical reservoir engineering deals with the reservoir on a gross average basis (tank model) and cannot account adequately for the variations in reservoir and fluid parameters in space and time. Reservoir simulation on computers allows a more detailed study of a reservoir by segmentation into a number of blocks and by applying fundamental equations of flow in porous media to each block. The governing flow equations describe fluid and pressure distributions in the porous medium (which represents the reservoir). The laws governing flow in porous media are based on the conservation of mass, momentum and energy and are discussed in detail in numerous books (Bird et al., 1960). From a practical standpoint, the semiempirical Darcy's law is broadly used in place of the momentum balance equation (Aziz and Settari, 1979). The hydrocarbon system of multiphase flow is usually restricted to the so-called black oil model where the system is approximated by three components, a nonvolatile component (black oil), a volatile component (gas) soluble in the oil phase and a water phase. In the black oil model, the oil-gas phase behavior is represented by formation factors and solution gas-oil ratio curves and additional mass

transfer of oil components from liquid to gas is not considered in the model (Aziz and Settari, 1979). The fluid approximations of this model are found to be acceptable for a large percentage of the world's oil reservoirs. Thus, black oil simulators have a wide range of applicability and are routinely used for solving field production problems (Chang et al., 1992).

Once the governing equations for a model of oil reservoir are set up, then efficient methods for solving the partial differential equations are sought. If the reservoir is defined using a geologic model, the fine detail is generated using a variety of geological data such as seismic information, well logs, and core interpretation. The model typically has several million grid cells. The numerical simulation, however, consolidates these grid blocks into several tens to several hundreds of upscaled model cells with effective reservoir properties. Fine resolution simulation requires considerable computational time. Hence, the first objective of this research effort was to apply the parallel, multiprocessor computing method to oil reservoir simulation.

Efficient parallel linear and nonlinear solvers lie at the core of the simulator since linear and nonlinear equations are solved several times for every time step during simulation. Thus, minor improvements in the efficiency of a solver will usually add up to significant improvements in the computational time for the entire simulation. With respect to linear solvers (examined in Chapter 2), the idea was to screen all of the modern linear solvers, both stationary and nonstationary, and select the one most appropriate for oil reservoir simulation. When a fully implicit scheme is used to provide unconditional stability to oil reservoir simulation, discretization of the partial differential equations result in a set of nonlinear algebraic equations to be solved at every time step. The most

popular choice for the solution of these nonlinear equations is the Newton's method. Although Newton's method can be applied in solving the equations, computing the exact solution using a direct method at each stage of the Newton's iteration, can be expensive in a simulation having a large number of unknowns. For reducing this computational cost, one of the project goals was to examine the inexact Newton's method developed recently (Eisenstat and Walker, 1996). The main idea behind this method is the inexact step, which saves the most computationally expensive part of computing the exact solution in Newton's iterations. The methods are also better streamlined for exploiting parallel implementation. Methods development and results are presented in Chapter 4.

Most oil reservoirs are fractured to a certain degree. The simulation of fluid flow through these naturally fractured oil reservoirs is a challenging problem due to the randomly shaped and distributed fractures. Research on fractured oil reservoir simulation has a history that spans nearly four decades. This involves development of a number of conceptual models describing flow in fractured media. Unfortunately, the models most currently used for fractured reservoir simulation do not explicitly consider the spatial fracture characterization. Instead, these conventional models smear out the fracture presence in either a dual continuum model (Kazemi, 1969) or a single continunm, equivalent property model (Watts, 1997). Conventional methods do not allow incorporation of fracture geometries and other attributes explicitly. Thus, the other primary project objective was to create a computational framework for the modeling of "discrete" fractures in reservoir simulation. In the discrete-fracture modeling approach, each individual fracture can be represented explicitly. A finite element formulation was chosen to handle the complex geometry of the discrete fracture system. Model

development and comparisons with the homogenization (upscaling) approaches are presented in Chapter 3. Solution of this complex geometric problem is also computationally intensive and appropriate parallel computing methods are desired, which is the topic of Chapter 5.

Although there have been a number of implementations of parallel computing in reservoir simulation (Killough, 1993 and Rame and Delshad, 1995), standards have not been established because the development has been machine specific and not portable. In this study, the Message Passing Interface (MPI) is used to provide the portability (Gropp et al., 1995). MPI, a standard portable message passing library was developed recently (1994) through collaboration of application scientists and people in the computer industry. MPI is a specification for a library of routines to be called from C or FORTRAN programs. Any FORTRAN or C code can be made parallel using this standard library of commands. In parallel computing, the memory that is required for the computational task may be shared among processors (shared memory computation) or could be distributed among different processors (distributed memory computation). Parallel implementation of the newly developed model was undertaken on shared memory and distributed-shared memory machines. Details of parallel implementation are the subjects of Chapter 2 (finite difference model) and Chapter 5 (finite element model) for regular and irregular grid systems.

Main objectives of this research are summarized below.

- To incorporate efficient linear solvers (using both the stationary and nonstationary methods) into reservoir simulation, particularly black oil reservoir simulation.
- To solve the nonlinear equations resulting due to the implicit formulation of the partial differential equations using the more advanced Inexact Newton's Krylov subspace

methods.

- To implement the above algorithms on parallel processors using standardized MPI protocol.
- To develop a discrete fracture, finite element, multiphase model using efficient nonlinear and linear equations solution algorithms.
- To test the finite element model on multiple processors.

CHAPTER 2

PARALLEL ITERATIVE LINEAR SOLVERS FOR OIL RESERVOIR SIMULATION

2.1 Introduction

Parallel computing methods applied to oil reservoir simulation improve the ability of the simulators in representing fine scale reservoir geology. One of the critical issues in the parallelization process is the solution of linear matrix systems arising from subdividing a single reservoir and assigning decomposed reservoir domains to multiple processors. This chapter presents a study of parallel iterative solutions of the linear matrix systems occurring in the parallel implementation of oil reservoir simulation. The matrices created by the discretization of the partial differential equations are sparse, unsymmetrical systems of equations, which can be written as:

$$Ap = b \tag{2.1}$$

Typical linear equations system in parallel oil reservoir simulation involves complex grids with hundreds of thousands of cells having one or more unknowns. Because the solution of these linear systems consumes vast computational resources,

efficient parallel linear solvers that provide speed, flexibility, and reliability are of great importance. Although modern direct solvers provide fast and robust solutions, this study is focused on iterative methods, which are substantially superior for a large number of linear equations. The iterative methods examined here were as follows:

- Stationary method

  Linear Successive Over Relaxation (LSOR) method (Chang et al., 1992)

- Nonstationary methods

  The stabilized version of the Bi-Conjugate Gradient (Bi-CGSTAB) method (Van Der Vorst, 1992)

  The Restarted Generalized Minimal Residual (RGMRES) method (Saad and Schultz, 1986)

  The Transpose Free Quasi Minimal Residual (TFQMR) method (Freund, 1993)

In solving the linear system of equation 2.1, iterative methods of the form, $p_{m+1} = M\, p_m + c$ are called stationary methods because the iteration from p_m to p_{m+1} does not depend on the history of the iteration. In the above equation, M is the iteration matrix. The stationary iterative solvers such as Jacobi and Gauss-Seidel line relaxation methods have the problem of pipelined solution on multiple processors. However, LSOR method in this study is based on the idea that the concurrent matrix solvers to avoid the cost of pipelining in parallel environment replace the sequential matrix solvers (Hofhaus and Van De Velde, 1996). The method is relatively simple to implement and provides reasonable parallel performance.

Conjugate-gradient-like methods (also known as Krylov subspace methods) are the most prominent nonstationary iterative techniques, where information of the iteration

matrix affecting computations changes at every iteration step. Unlike the stationary iterative methods, Krylov subspace methods minimize some measured error over a space $p_{initial} + k$, where $p_{initial}$ is the initial iterate and k is the Krylov subspace. An appropriate preconditioning procedure is essential for the success of these methods. Based on effectiveness and simplicity, the Gauss-Seidel preconditioner was chosen in this study. In this chapter, a brief overview of the iterative methods is presented and the parallel performance of iterative linear solvers is studied for oil reservoir simulation. The application is three-dimensional and three-phase black oil simulation. The equations were discretized by finite difference method using implicit pressure and explicit saturation (IMPES) scheme (Aziz and Settari, 1979 and Chang et al., 1992).

2.2 Governing equations

The differential equations and the subsequent difference equations, which can be derived combining the material balance with Darcy's law, are given by:

$$\text{Oil phase: } \nabla \cdot \left(T_o \nabla \left(P_o - g_{\ o} Z\right)\right) = \frac{\partial}{\partial t}\left(f \frac{S_o}{B_o}\right) + \frac{q_o}{r_{osc}} \tag{2.2}$$

$$\text{Water phase: } \nabla \cdot \left(T_w \left(\nabla P_w - g_{\ w} Z\right)\right) = \frac{\partial}{\partial t}\left(f \frac{S_w}{B_w}\right) + \frac{q_w}{r_{osw}} \tag{2.3}$$

Gas phase:

$$\nabla \cdot \left(T_g \nabla \left(P_g - g_{\ g} Z\right)\right) + \nabla \cdot \left(R_{so} T_o \nabla \left(P_o - g_{\ o} Z\right)\right) = \frac{\partial}{\partial t}\left(f \frac{S_g}{B_g} + f R_{so} \frac{S_o}{B_o}\right) + \frac{q_g}{r_{gsc}} + R_{so} \frac{q_o}{r_{osc}} \tag{2.4}$$

Capillary pressures: $P_{cow} = P_0 - P_w,\ P_{cgo} = P_g - P_o$ (2.5)

Phase saturation: $S_o + S_w + S_g = 1$ (2.6)

In these equations, P_l (pressure of phase l) and S_l (saturation of phase l) are the primary dependent variables to be solved for each phase. The idea of IMPES scheme is to derive a single pressure equation by combining the flow equations 2.2, 2.3, 2.4, 2.5 and 2.6. Once the pressure equation is solved implicitly, the saturations are explicitly updated by substituting the results in the flow equations 2.2, 2.3 and 2.4. The pressure equation is written as (Aziz and Settari, 1979 and Chang et al., 1992):

$$(B_o - R_{so} B_g)[\nabla \cdot T_o(\nabla P_o - \nabla g_o Z) - \frac{q_o}{r_{osc}}] + B_w[\nabla \cdot T_w(\nabla P_o - \nabla P_{cow} - \nabla g_w Z) - \frac{q_w}{r_{wsc}}] +$$
$$B_g[\nabla \cdot (T_g + R_{so} T_o)\nabla P_o + \nabla \cdot T_g(\nabla P_{cgo} - \nabla g_g Z) - \nabla \cdot R_{so} T_g(\nabla g_o Z) - \frac{q_g}{r_{gsc}}] = f c_t \frac{\partial P_o}{\partial t} \quad (2.7)$$

In this equation, the total compressibility c_t is defined as:

$$c_t = (-\frac{1}{B_o}\frac{\partial B_o}{\partial P_o} + \frac{B_g}{B_o}\frac{\partial R_{so}}{\partial P_o})S_o - \frac{S_w}{B_w}\frac{\partial B_w}{\partial P_o} - \frac{S_g}{B_g}\frac{\partial B_g}{\partial P_o} \quad (2.8)$$

The finite difference form of the pressure equation can be expressed in the following seven-point stencil.

$$AE_i P_{i+1} + AW_i P_{i-1} + AN_j P_{j+1} + AS_j P_{j-1} + AB_k P_{k+1} + AT_k P_{k-1} + E_{i,j,k} P_{i,j,k} = b_{i,j,k} \quad (2.9)$$

which has the matrix form of equation 2.1 ($Ap=b$). The coefficients are:

$$AE_i = [B_o + \frac{1}{2} B_g (R_{so,i+1} - R_{so,i})]\tilde{T}_{o,i+\frac{1}{2}} + B_w \tilde{T}_{w,i+\frac{1}{2}} + B_g \tilde{T}_{g,i+\frac{1}{2}} \quad (2.10a)$$

$$AW_i = [B_o + \frac{1}{2} B_g (R_{so,i-1} - R_{so,i})]\tilde{T}_{o,i-\frac{1}{2}} + B_w \tilde{T}_{w,i-\frac{1}{2}} + B_g \tilde{T}_{g,i-\frac{1}{2}} \quad (2.10b)$$

$$AN_j = [B_o + \frac{1}{2} B_g (R_{so,j+1} - R_{so,j})]\tilde{T}_{o,j+\frac{1}{2}} + B_w \tilde{T}_{w,j+\frac{1}{2}} + B_g \tilde{T}_{g,j+\frac{1}{2}} \quad (2.10c)$$

$$AS_j = [B_o + \frac{1}{2} B_g (R_{so,j-1} - R_{so,j})]\tilde{T}_{o,j-\frac{1}{2}} + B_w \tilde{T}_{w,j-\frac{1}{2}} + B_g \tilde{T}_{g,j-\frac{1}{2}} \quad (2.10d)$$

$$AB_k = [B_o + \frac{1}{2} B_g (R_{so,k+1} - R_{so,k})]\tilde{T}_{o,k+\frac{1}{2}} + B_w \tilde{T}_{w,k+\frac{1}{2}} + B_g \tilde{T}_{g,k+\frac{1}{2}} \quad (2.10e)$$

$$AT_k = [B_o + \frac{1}{2} B_g (R_{so,k-1} - R_{so,k})]\tilde{T}_{o,k-\frac{1}{2}} + B_w \tilde{T}_{w,k-\frac{1}{2}} + B_g \tilde{T}_{g,k-\frac{1}{2}} \quad (2.10f)$$

$$E = -[AE_i + AW_i + AN_j + AS_j + AB_k + AT_k + \frac{V_p^n c_t^n}{\Delta t}] \quad (2.10g)$$

$$b = -[-QOWG + \frac{V_p^n c_t^n p^n}{\Delta t}] \quad (2.10h)$$

$$QOWG = (B_o - B_g R_{so})(\frac{q_o V_B}{r_{osc}} - GOWT) + B_w(\frac{q_w V_B}{r_{wsc}} - GWWT) + B_g(\frac{q_g V_B}{r_{gsc}} - GGWT) \quad (2.10i)$$

$$QOWT = -\nabla \cdot \tilde{T}_o^n \nabla(g_o Z)^n \quad (2.10j)$$

$$QWWT = -\nabla \cdot \tilde{T}_w^n \nabla(g_w Z + P_{cow})^n \quad (2.10k)$$

$$QGWT = \nabla \cdot [\tilde{T}_g^n \nabla(p_{cgo} - g_w Z)^n - R_{so} \tilde{T}_o^n \nabla(g_o Z)^n] \quad (2.10l)$$

Here, the differential operator is defined as:

$$\nabla \cdot \tilde{T}_l \nabla P_l = \frac{2\Delta x_i}{(x_{i+1} - x_{i-1})}[T_{l,i-\frac{1}{2},j,k}(p_{i-1,j,k} - p_{i,j,k}) + T_{l,i=\frac{1}{2},j,k}(p_{i+1,j,k} - p_{i,j,k})] +$$

$$\frac{2\Delta y_i}{(y_{i+1} - y_{i-1})}[T_{l,i,j-\frac{1}{2},k}(p_{i,j-1,k} - p_{i,j,k}) + T_{l,i,j-\frac{1}{2},k}(p_{i,j+1,k} - p_{i,j,k})] + \quad (2.11)$$

$$\frac{2\Delta z_i}{(z_{i+1} - z_{i-1})}[T_{l,i,j,k-\frac{1}{2}}(p_{i,j,k-1} - p_{i,j,k}) + T_{l,i,j,k=\frac{1}{2}}(p_{i,j,k+1} - p_{i,j,k})]$$

The phase (*l*) transmissibility between grid block *i-1* and grid block *i* is given by:

$$T_{l,i-\frac{1}{2},j,k} = \frac{8k_{rl}(k_{absolute}Ac)_{i-1}(k_{absolute}Ac)_i}{\{\Delta x_{i-1}(k_{absolute}Ac)_i + \Delta x_i(k_{absolute}Ac)_{i-1}\}(m_{l,i-1} + m_{l,i})(B_{l,i-1} + B_{l,i})} \quad (2.12)$$

Note that the finite difference transmissibility of the phase l $\tilde{T}_l$ was defined as:

$$\tilde{T}_{l,i-\frac{1}{2},j,k} = \frac{2\Delta x_i}{(x_{i+1} - x_{i-1})} T_{l,i-\frac{1}{2},j,k} \quad (2.13)$$

2.3 Parallel domain decomposition

Domain decomposition method has conventionally been used to solve the linear matrix system of equation 2.9 ($Ap=b$) on multiple processors. In this approach, the grid cells of the reservoir system are subdivided onto different processors. Then, the data communication occurs for the boundary grid cells, which are dependent on the grid cells of other processors. Examining the structure of regular grid system used in finite

difference method, the domain can be decomposed using three different techniques as follows.

- One-dimensional partitioning: The data set is partitioned along columns and the parallel communication is required in a single direction.
- Two-dimensional partitioning: The data set is partitioned along rows and columns, also known as panel partitioning. Data communication is required on two faces of a three-dimensional box.
- Three-dimensional partitioning: In this method, the data set is partitioned along all the faces of the three-dimensional box. The data communication then occurs in all the three directions.

The three methods are illustrated in Figure 2.1. The simplest of the approach is the one-dimensional decomposition scheme and the most complex is the three-dimensional domain decomposition method. A dimensionality of domain decomposition increases so does the communication requirement (Figure 2.1). Once the problem domain is decomposed on multiple processors, each processor is responsible for computations in its respective domain. A processor communicates with others only if data is nonlocal. For example, Figure 2.2 shows a three-dimensional finite difference domain consisting a $3 \times 2 \times 2$ grid system which is split by a horizontal plane (x-y) into two processors. The first horizontal layer is assigned to the first processor and the second layer is allocated to the second processor. Processor I will be required to obtain the information at the grid cells, (1~3, 1~2, 2), and Processor II needs to update information at points, (1~3, 1~2, 1). Resulting from the domain decomposition, the subsequent structure of linear matrix system of equation 2.9 is also shown in Figure 2.2. The domain

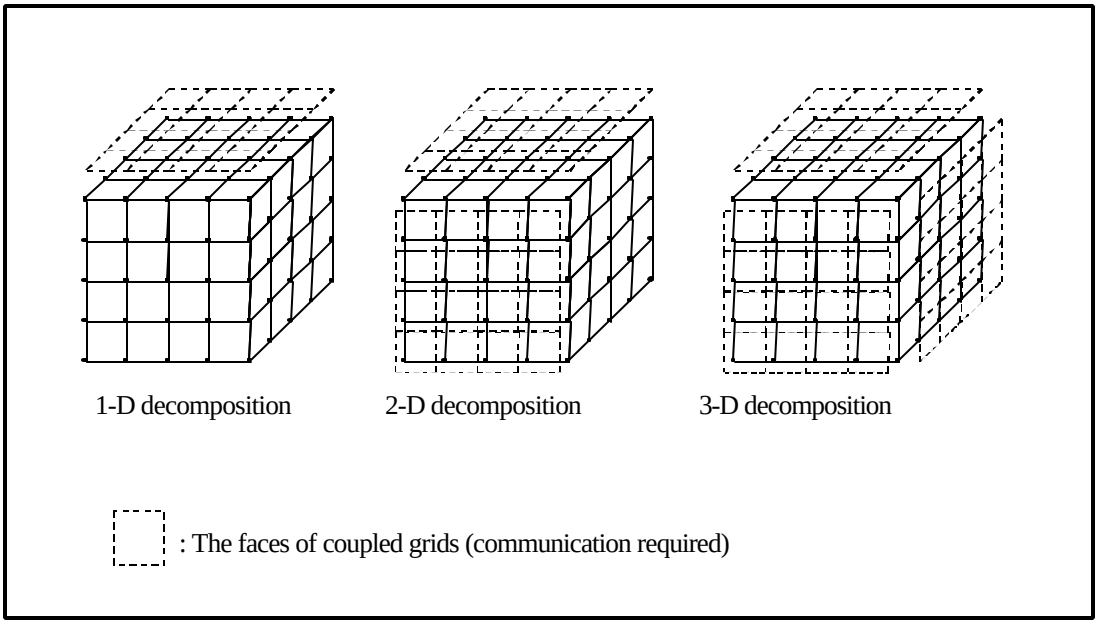


Figure 2.1 The domain decomposition schemes for multiprocessor reservoir simulation

decomposition in z-direction attributes the elements $AB_{i,j,k}$ and $AT_{i,j,k}$ coupled in two processors. Thus, all the information to make these matrix elements is supposed to be exchanged in the two processors. Resulting from the parallel domain decomposition, the parallel communication required in the operations of linear matrix system distributed over multiple processors is discussed in the following section. MPI library provides the tools necessary to assign each processor to the decomposed subdomains of regular grid systems for better performance. MPI tools are provided in the Appendix A.

2.4 Overview of parallel linear solvers

The study chooses four representative linear solvers for both stationary and nonstationary iterative methods; LSOR, Bi-CGSTAB, RGMRES and TFQMR methods. Basic features and algorithms of these methods are well known in literatures (Press et al.,

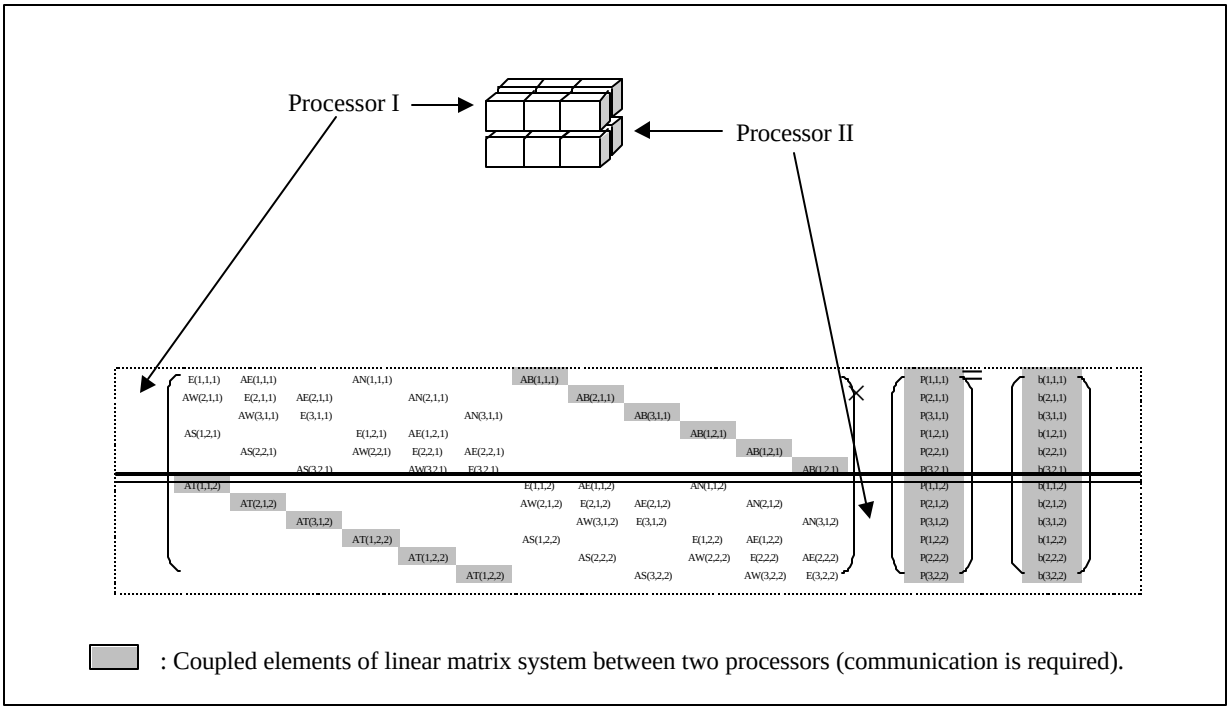


Figure 2.2 The structure of the heptadiagonal matrix for a 3×2×2system that has been divided among two processors.

1992, Van Der Vorst, 1992, Saad and Schultz, 1986 and Freund, 1993). Additional to a brief overview of each linear solver, the following subsections address the issues involved in conversion of the iterative methods to parallel codes.

2.4.1 Stationary LSOR method on multiple processors

To solve the linear matrix system of equation 2.9 ($A\,p = b$), the LSOR method used in this study are started by splitting the coefficient matrix A (here, heptadiagonal matrix) into

$$A = G + H \tag{2.14}$$

Here G is a matrix that can easily be inverted. This defines the iteration

$$Gp_m = b - Hp_{m-1} \tag{2.15}$$

where P_m converges to the exact solution p^* and P_{m-1} is the pressure value from previous iteration. For example, if a x-line LSOR is a set of grid points (i, j, k) where $1 \le i \le I$, $1 \le j \le J$ and $1 \le k \le K$, the linear matrix system of equation 2.9 will reduce to the tridiagonal system

$$AW_{i,j,k}\, p_{i-1,j,k} + E_{i,j,k} p_i + AE_{i,j,k}\ p_{i+1} = b_{i,j,k} - Hp_{i,j,k} \tag{2.16}$$

In matrix-vector notation, this system has the form

$$Gp = b - Hp_{m-1} \tag{2.17}$$

with

$$G = \begin{pmatrix} E_{1,1,1} & AE_{1,1,1} & & & & & \\ AW_{2,1,1} & E_{2,1,1} & AE_{2,1,1} & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & AW_{I-1,J,K} & E_{I-1,J,K} & AE_{I-1,J,K} \\ & & & & & AW_{I,J,K} & E_{I,J,K} \end{pmatrix}$$

Here, the matrix H contains the other elements of the heptadiagonal matrix system; $AN_{i,j,k}$, $AS_{i,j,k}$, $AT_{i,j,k}$ *and* $AB_{i,j,k}$. The advantage of the tridiagonal system of equations is that there are a number of solver subroutines available for such systems in computational libraries like LAPACK. For an example of tridiagonal system on multiple processors, suppose that a domain ($I = 6$) is partitioned in i or x direction such that half of the data nodes are assigned to one processor and the other half are assigned to a second processor. The data distribution over two processors in the x-direction forces us to solve the following distributed tridiagonal matrix system for the x-line LSOR method.

$$\left(\begin{array}{ccccccc} E_1 & AE_1 & & \text{Processor I} & & & \\ AW_2 & E_2 & AE_2 & & & & \\ & AW_3 & E_3 & AE_3 & & & \\ \hline & & AW_4 & E_4 & AE_4 & & \\ & & & AW_5 & E_5 & AE_5 & \\ & \text{Processor II} & & & AW_6 & E_6 & \end{array}\right) \times \left(\begin{array}{c} P_1 \\ P_2 \\ P_3 \\ \hline P_4 \\ P_5 \\ P_6 \end{array}\right) = \left(\begin{array}{c} b_1 \\ b_2 \\ b_3 \\ \hline b_4 \\ b_5 \\ b_6 \end{array}\right) \tag{2.18}$$

If the classical sequential solver (referred to as Thomas algorithm) is used to solve the above matrix system, the structure of the system indicates that processor I (AE_3) requires a data (P_4) from processor II and processor II (AW_4) is required to wait until processor I updates the data (P_3). Thus, processor II is an idle processor while processor I sweeps its own domain in a sequential algorithm. As an alternative to this pipelining solution scheme, Hofhaus and Van De Velde (1996) suggested an iterative method to replace the sequential tridiagonal solver by a concurrent tridiagonal solver. For the purpose of concurrent parallel computing, the method constructs tridiagonal systems independent of one another so that they can be solved simultaneously. The new matrix systems can be obtained where matrix G contains only globally uncoupled tridiagonal blocks. All the other coefficients crossing process boundaries are carried over to matrix H, which may additionally contain the matrix coefficients split from the x-line LSOR scheme. This requires the linear matrix system of equation 2.18 to be split $G_{uncoupled} + H_{coupled}$, as follows:

$$\left(\begin{array}{cccc|cccc} E_1 & AE_1 & & & & & & \\ AW_2 & E_2 & AE_2 & & & & & \\ & AW_3 & E_3 & & & & & \\ \hline & & & E_4 & AE_4 & & & \\ & & & AW_5 & E_5 & AE_5 & & \\ & & & & AW_6 & E_6 & & \end{array}\right) \times \left(\begin{array}{c} P_1 \\ P_2 \\ P_3 \\ \hline P_4 \\ P_5 \\ P_6 \end{array}\right)_{\mathrm{m}} = \left(\begin{array}{c} b_1 \\ b_2 \\ b_3 \\ \hline b_4 \\ b_5 \\ b_6 \end{array}\right) - \left(\begin{array}{cc} & AE_3 \\ \hline AW_4 & \end{array}\right) \times \left(\begin{array}{c} P_3 \\ \hline P_4 \end{array}\right)_{\mathrm{m-1}} \qquad (2.19)$$

This splitting of the coefficient matrix defines the iteration

$$G_{uncoupled} p_m = b - H_{coupled} p_{m-1} \qquad (2.20)$$

A parallel implementation of the iteration is easily obtained as follows:

1. G contains only uncoupled matrices and its inversion is trivially concurrent.
2. The parallel matrix-vector operation $b - H_{coupled}\, p_{m-1}$ is performed with only the nearest-neighbor communication.
3. The solution of the local linear system of equations is started independently on multiple processors.
4. The right-hand side terms are updated by communicating the first and the last row in each process.
5. The iteration is continued until some stopping criterion is satisfied.

Hofhaus and Van De Velde (1996) suggested this method. With this iterative method, the concurrent solution scheme of LSOR method can be implemented on multiple processors. Since all the processors start working on the system of equations simultaneously, there are no idle processors. In the parallel implementation, the updated pressure values (P_3 and P_4) of equation 2.19 can be communicated using MPI library as follows.

```
include 'mpif.h'
integer status(MPI_STATUS_SIZE,4),er(4)
Parameter (ni=3)
common /Prtp/ P(0:ni+1)
integer top,bottom
common /neigh/ top,bottom
call MPI_ISEND(p(1),1,MPI_REAL,top,0,MPI_COMM_WORLD,er(1),info)
call MPI_ISEND(p(3),1,MPI_REAL,bottom,0,MPI_COMM_WORLD,er(2),info)
call MPI_IRECV(p(0),1,MPI_REAL,top,0,MPI_COMM_WORLD,er(3),info)
call MPI_IRECV(p(4),1,MPI_REAL,bottom,0,MPI_COMM_WORLD,er(4),info)
call MPI_WAITALL(4,er,status,info)
```

This communication must occur before each processor performs the next iteration. The pseudo code for the method is given in Figure 2.3. The basic steps involved in using MPI tools are provided in the Appendix A.

2.4.2 Nonstationary parallel Krylov subspace methods

The idea of conjugate gradient iteration was invented in the 1950s (Hestenes and Stiefel, 1952 and Fletcher, 1976). The basic process in each of the conjugate-gradient-like methods is the projection of the original set of equations onto a subspace (the so-called Krylov subspace). In the projection onto Krylov subspace, the following quadratic function is minimized by the solution of equation 2.1 or the solution of pressure equation 2.9 derived by IMPES scheme in this study.

$$F(p) = \frac{1}{2} p^T A p - b^T p + c \tag{2.21}$$

To solve linear matrix system Ax = b.

1. x_o is an initial guess
2. Set $A = G + H$

For k = 1, 2, …

3. Solve $G\, x_k = b - H\, x_{k-1}$
4. Check stopping criterion

Endfor

Figure 2.3 The LSOR method

In the minimization of equation 2.21, a Krylov subspace method seeks an approximate solution p_m from different choices of Krylov subspaces of the form

$$k^m = span\{r_0, Ar_0, A^2 r_0, ..., A^{m-1} r_0\} \tag{2.22}$$

in which approximate solution is searched by a vector of the form

$$p_m = p_0 + polynomial\ (A) r_0 \tag{2.23}$$

Among different versions of the Krylov methods, the study examined parallel performance of minimal residual methods (RGMRES and TFQMR) and bi-conjugate gradient method (Bi-CGSTAB) in solving the linear matrix system of equation 2.9.

In the family of minimal residual methods, the generalized minimal residual method (GMRES) is the most popular. At the start, a GMRES is much like any conjugate-gradient method with the residuals forming an orthogonal basis for the Kylov subspace in the form of $r_m = Polynomial(A) r_o$. For the implementation of GMRES in the study, the so-called Anoldi orthogonalization method (Kelley, 1995) is used to project an approximate solution vector in the Krylov subspace. But the method is storage intensive to retain all previously computed vectors of orthogonal projection onto the Krylov subspace. For this reason, restarted version of the method is often employed. In the following section, GMRES*(m)* refers to GMRES restarted after every m iterations. The pseudocode for the restarted GMRES (m) algorithm with preconditioner Q_1 is given in Figure 2.4.

To solve linear matrix system Ax = b.

1. x_o is an initial guess
2. $r_o = Q_1(b - Ax_o)$
3. $b_o = \|r_o\|$

   for k = 1, 2, …
4. $g = (b_{k-1}, b_{k-1}, \cdots)^T$
5. $V_1 = r_{k-1} / b_{k-1}$

   for j = 1, 2, …, m
6. $R_{i,j} = V_i^T Q_1 A V_j, i = 1, \ldots, j$
7. $\bar{u} = Q_1 A V_j - \sum_{i=1}^{j} R_{i,j} V_i$
8. $R_{j+1,j} = \|\bar{u}\|$
9. $V_{j+1} = \bar{u} / R_{j+1,j}$
10. apply previous Given's rotations to $R_{\cdot,j}$
11. compute Given's rotation to zero $R_{j+1,j}$
12. apply Given's rotation to g
13. check stopping criterion to g

    endfor
14. solve $Ry = g$
15. $x_k = x_{k-1} + Vy$ form iterative solution
16. $r_k = Q_1(b - Ax_k)$
17. $b_k = \|r_k\|$

    endfor

Figure 2.4 The preconditioned GMRES(*m*) method

When the residual polynomial reduces r_o to a smaller vector r_m in the Krylov subspace methods, the bi-conjugate-gradient-stabilized method (Bi-CGSTAB) is based on the idea of further contraction of this operation. Bi-CGSTAB applies the contraction polynomial twice using the polynomial describing the steepest descent update. This further contraction can lead a considerably smoother convergence behavior at least

locally. However, if the Krylov subspace is not expanded, and Bi-CGSTAB will breakdown. The psuedocode for the method with preconditioner Q_1 is given in Figure 2.5.

While the Bi-CGSTAB method often displays irregular convergence behavior, another related algorithm, the transpose-free-quasi-minimal-residual method (TFQMR) attempts to overcome the problem. The main idea behind this algorithm is to minimize the norm of a quasi-residual, which can be related to the true residual by a linear

To solve linear matrix system Ax = b.

1. x_o is an initial guess
2. $r_o = Q_1(b - Ax_o)$
3. $\bar{r}_o = r_o$
4. $p_o = u_o = 0$
5. $r_o = a_o = w_o = 1$

   for k = 1, 2, …
6. $r_k = \bar{r}_o^T r_{k-1}$
7. $b_k = r_k a_{k-1} / (r_{k-1} w_{k-1})$
8. $p_k = r_k + b_k (p_{k-1} - w_{k-1} u_{k-1})$
9. $u_k = Q_1 A p_k$
10. $x_k = \bar{r}_o u_k$
11. $s_k = r_{k-1} - a_k u_k$
12. if $\|s_k\| \leq$ tolerance, breakdown is occurred.
13. $t_k = Q_1 A s_k$
14. $w_k = t_k^T s_k / t_k^T t_k$
15. $x_k = x_{k-1} + a_k p_k + w_k s_k$
16. $r_k = s_k - w_k t_k$
17. check stopping criterion

    endfor

Figure 2.5 The preconditioned BiCGSTAB method

transformation. In addition to the idea of quasi-minimal-residual minimization, TFQMR uses look-ahead techniques to avoid breakdowns due to the indefiniteness of underlying vectors in searching the solution vector of equation 2.21. The psuedocode for the method with preconditioner Q_1 is given in Figure 2.6.

When Krylov subspace methods are implemented on multiple processors, parallel communications involving the following basic matrix operations are required.

- Matrix vector multiplications
- Updating vectors
- Dot products of vectors
- Operations of local preconditioner

Matrix and vector operations of the linear matrix system of equation 2.9 are organized as follows. The domain configuration of Figure 2.2 is taken for an example. To compute matrix-vector product, each processor exchanges with its neighbor the grid points in the interface of the processors. In Figure 2.2, *AB(1~3,1~2,1)* of processor I requires the *P(1~3,1~2,1)* from processor II and *AT(1~3,1~2,2)* of processor II needs to obtain the *P (1~3,1~2,1)* from processor II. Equation 2.9 is then applied independently by each processor at its local grid points, except at the local interfacing points. After the interfacing grid points from neighboring processors have arrived at each processor. Equation 2.9 is fully applied using the interfacing points from the neighboring processors. The example below is taken for a matrix-vector product routine using MPI library.

```
include 'mpif.h'
integer status(MPI_STATUS_SIZE,4),er(4)
Parameter (ni=3, nj=2, nk=1)
```

To solve linear matrix system Ax = b.

1. x_o is an initial guess
2. $r_o = Q_1(b - Ax_o)$
3. $\bar{r}_o = r_o = w_1 = y_1 = r_o$
4. $g = v_o = Q_1 A y_1$
5. $d_o = q_o = h_o = 0$
6. $t_o = \|r_o\|$

   for k = 1, 2, …
7. $s_{k-1} = \bar{r}_o^T v_{k-1}$
8. $a_{-1k} = r_{k-1} / s_{k-1}$
9. $h = Q_1 A y_{2k}$

   for m = 2k-1, 2k
10. $w_{m+1} = w_m - a_{k-1} g$
11. $q_m = \|w_{m+1}\| / t_{m-1}$
12. $c_m = 1/\sqrt{1+q_m}$
13. $t_m = t_{m-1} q_m c_m$
14. $h_m = c_m^2 a_{k-1}$
15. $t_k = Q_1 A s_k$
16. $d_m = y_m + \left(q_{m-1}^2 h_{m-1} / a_{k-1}\right) d_{m-1}$
17. $x_m = x_{m-1} + h_m d_m$
18. check stopping criterion
19. $g \leftarrow h$

    endfor
20. $r_k = \bar{r}_o^T w_{2k+1}$
21. $b_k = r_k / r_{k-1}$
22. $y_{2k+1} = w_{2k+1-1} + b_k y_{2k}$
23. $g = Q_1 A y_{2k+1}$
24. $v_k = g + b_k (h + b_k v_{k-1})$

    endfor

Figure 2.6 The preconditioned TFQMR method

```
 common /Prtp/ P(0:ni+1,0:nj+1,0:nk+1)
 COMMON /COEF/ AW(ni,nj,nk), AE(ni,nj,nk), AN(ni,nj,nk), AS(ni,nj,nk),
&             AB(ni,nj,nk), AT(ni,nj,nk),  E(ni,nj,nk),  B(ni,nj,nk)
 integer top,bottom
 common /neigh/ top,bottom
call MPI_TYPE_VECTOR(nj+2,ni+2,ni+2,MPI_REAL,k_type,info)
call MPI_TYPE_COMMIT(k_type,info)
call MPI_ISEND(p(0,0,1),1,k_type,top,0,MPI_COMM_WORLD,er(1),info)
call MPI_ISEND(p(0,0,nk),1,k_type,bottom,0,MPI_COMM_WORLD,er(2),info)
call MPI_IRECV(p(0,0,0),1,k_type,top,0,MPI_COMM_WORLD,er(3),info)
call MPI_IRECV(p(0,0,nk+1),1,k_type,bottom,0,MPI_COMM_WORLD,er(4),info)
call MPI_WAITALL(4,er,status,info)
 do k=1,nk
 do j=1,nj
 do i=1,ni
 IM=i-1
 IP=i+1
 JM=j-1
 JP=j+1
 KM=k-1
 KP=k+1
 b(i,j,k)=AW(i,j,k)*P(IM,j,k) + AE(i,j,k)*P(IP,j,k)
&         AN(i,j,k)*P(i,JP,k) + AS(i,j,k)*P(i,JM,k)
&         AB(i,j,k)*P(i,j,KP) + AT(i,j,k)*P(i,j,KM)
&          E(i,j,k)*P(i,j,k)
 enddo
 enddo
 enddo
```

```
call MPI_TYPE_FREE(k_type,info)
```

The other computations of matrices and vectors are performed in a similar fashion. Exchanging information of grid cells with neighboring processors were satisfied by communication functions in MPI library allowing the local matrix and vector operations to be implemented in a conventional fashion. For more detail of MPI tools, refer to the Appendix A.

2.4 Numerical results

The parallel performance of the program was tested on a typical black-oil three-dimensional problem (Table 2.1). Relative permeabilities and capillary pressures used in test simulations are shown in Table 2.2. Other basic properties (viscosities, oil and gas formation volume factors, gas oil ratio, etc) are provided in Table 2.3. To measure parallel performance, speedup was calculated as follows.

$$Speed\ up_{(n)} = \text{cpu time on 1 processor / cpu time on } n \text{ processors} \qquad (2.24)$$

For the test run, $p_{initial} = 0$ is always chosen as initial guess. The iteration was then stopped when the converge criterion, $||A\ p_k - d|| / ||A\ p_o - d|| = 10^{-5}$ was satisfied. Three restarts were used for GMRES, since the work and storage per iteration were roughly comparable to other methods.

The convergence behavior of the four parallel linear solution methods is shown in Figure 2.7. A black-oil model with a total of 128,000 grid blocks ($80 \times 80 \times 20$) was chosen for this numerical experiment. The BiCGSTAB shows divergent behavior.

Table 2.1 Description of black oil model problem

Grid spacing,	delta x, m	3
	delta y, m	3
	delta z, m	6
Absolute permeability,	k_{xx}, mD	200
	k_{yy}, mD	200
	k_{zz}, mD	200
Porosity, fraction		0.2
Initial pressure, atm		102
Initial saturations,	S_o	0.8
	S_w	0.2
Bubble point pressure, atm		68
Stock tank oil density, g/cm^3		0.85
Stock tank water density, g/cm^3		1.0
Stock tank gas density, g/cm^3		0.001

Table 2.2 PVT properties for black oil model problem

Pressure Atm	Bo RC/STC	Bw RC/STC	Bg RCF/SCF	Solution Gas SCF/STC	μ_o cp	μ_w cp	μ_g cp
1	1.022	1.00	1.053	1.4	8.8	1.0	0.0108
24	1.064	1.00	0.027	89	4.6	1.0	0.0116
68	1.118	1.00	0.012	203	2.9	1.0	0.0133
102	1.177	1.00	0.007	329	2.1	1.0	0.0162
136	1.240	1.00	0.005	464	1.6	1.0	0.0201
170	1.306	1.00	0.004	604	1.3	1.0	0.0243
204	1.375	1.00	0.003	751	1.1	1.0	0.0281

Table 2.3 Relative permeabilities and capillary pressures employed for black oil model problem

Saturation	K_{ro}	K_{rw}	K_{rg}	P_{cow}atm	P_{cgo}atm
0.0	0.00	0.00	0.000	31	31
0.1	0.00	0.00	0.000	24	24
0.2	0.00	0.00	0.003	15	15
0.3	0.10	0.10	0.009	7	7
0.4	0.20	0.20	0.020	3	3
0.5	0.40	0.40	0.100	0.50	0.50
0.6	0.60	0.60	0.300	0.45	0.45
0.7	0.80	0.80	0.750	0.27	0.27
0.8	0.90	0.90	1.000	0.27	0.27
1.0	1.00	1.00	1.000	0.27	0.27

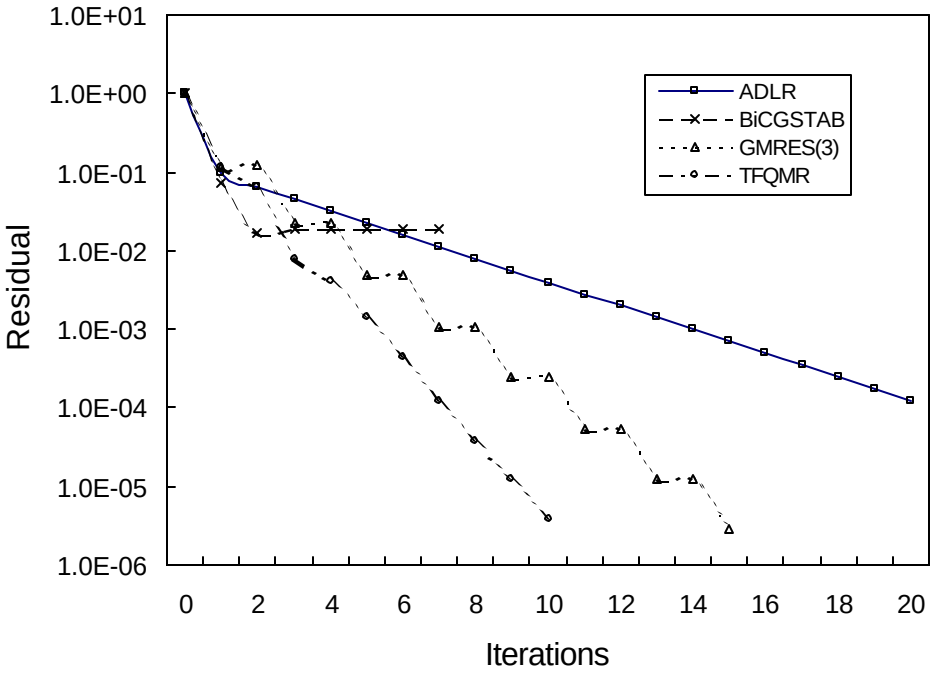


Figure 2.7 Convergence curves of iterative linear solvers for problem of (80,80,20) grids

TFQMR and GMRES*(3)* show the most rapid convergence. When multiple processors are used, the parallel performance of linear solvers may be compromised. To test this, a $36 \times 80 \times 20$ grid was employed in the study of parallel performance of each linear solver. Parallel TFQMR was tested on multiple processors. Figure 2.8 shows the parallel convergence behavior of TFQMR method on SGI Power Challenge. The convergence does slow down with multiple processors; however, the slow down is not significant. As the number of processors (nodes) increase, the performance gets only marginally worse, even though considerably more iterations are required when two processors are employed instead of one. The total speedup on multiple processors for three best parallel linear solvers (found in this study) is shown Figures 2.9, 2.10 and 2.11 where the composite computational time, and computational time for linear solver and matrix construction are shown. The overall performances of TFQMR and LSOR are comparable with TFQMR being marginally better. However, LSOR parallel implementation is much simpler than TFQMR. The parallel performance was also tested on a distributed memory machine (IBM SP2). The overall parallel performances and speedup for linear solution and matrix construction are shown in Figures 2.12, 2.13 and 2.14. The matrix computation speedup contributes more to the composite speedup in the case of the distributed memory machine in comparison to the shared memory machine. Different types of domain decomposition schemes were also studied on the shared memory SGI Power Challenge. The one-dimensional domain decomposition, in general, provided better speedup than either two- or three-dimensional divisions shown in Table 2.4

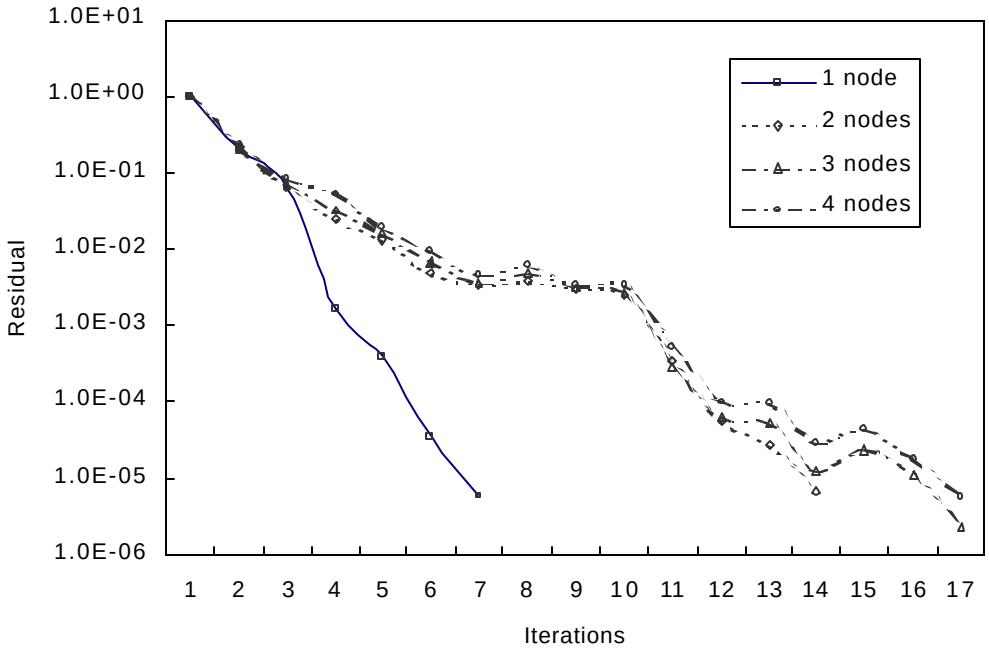


Figure 2.8 Convergence curves of TFQMR method for test problem of (36,80,20) grids

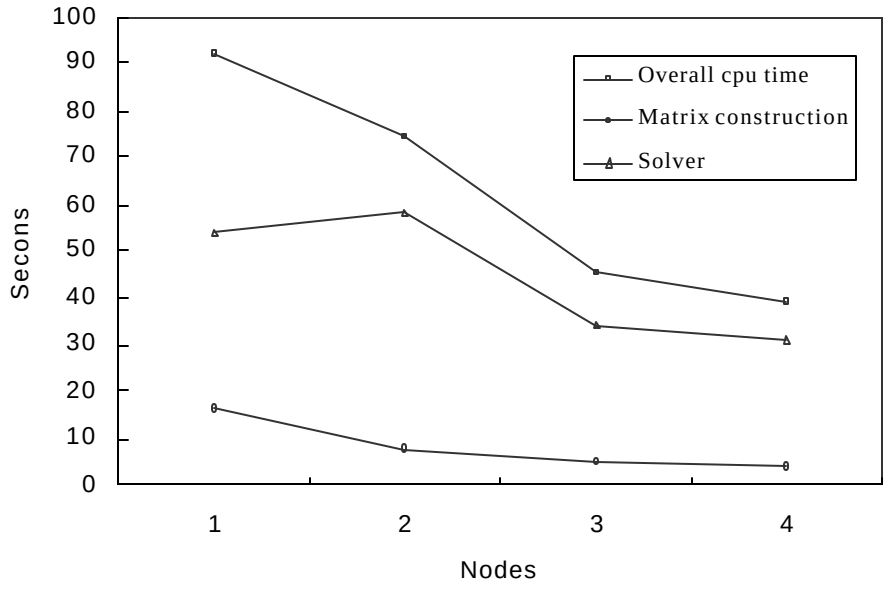


Figure 2.9 Speedup of LSOR on shared memory machine, SGI Power Challenge : (36,80,20) grids

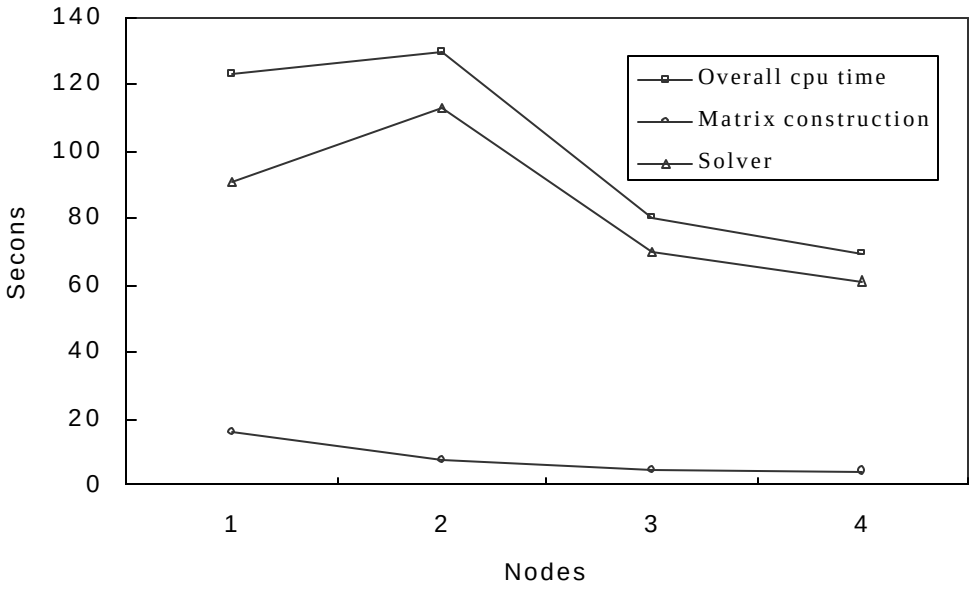


Figure 2.10 Speedup of GMRES(3) on shared memory machine, SGI Power Challenge : (36,80,20) grids

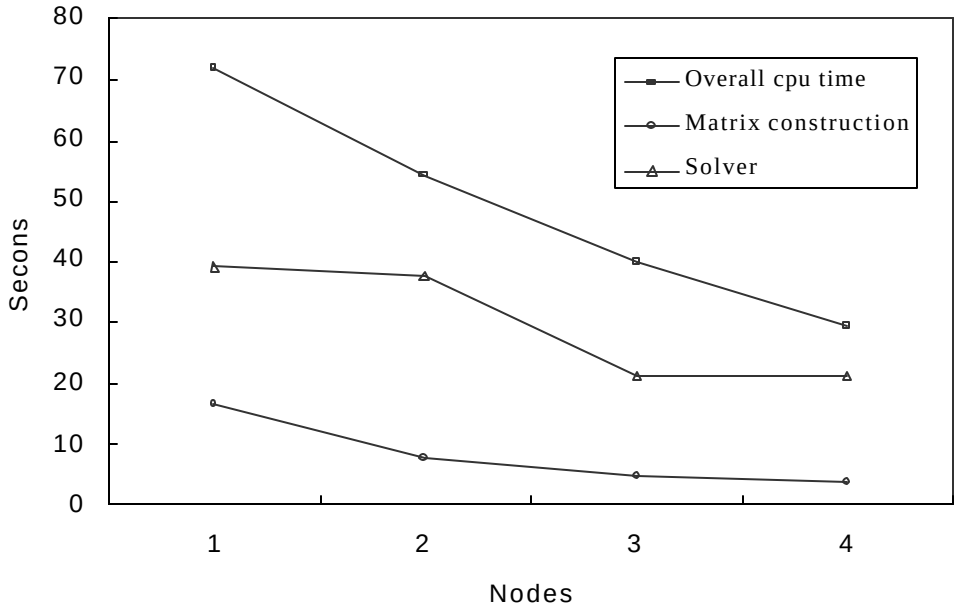


Figure 2.11 Speedup of TFQMR on shared memory machine, SGI Power Challenge: (36,80,20) grids

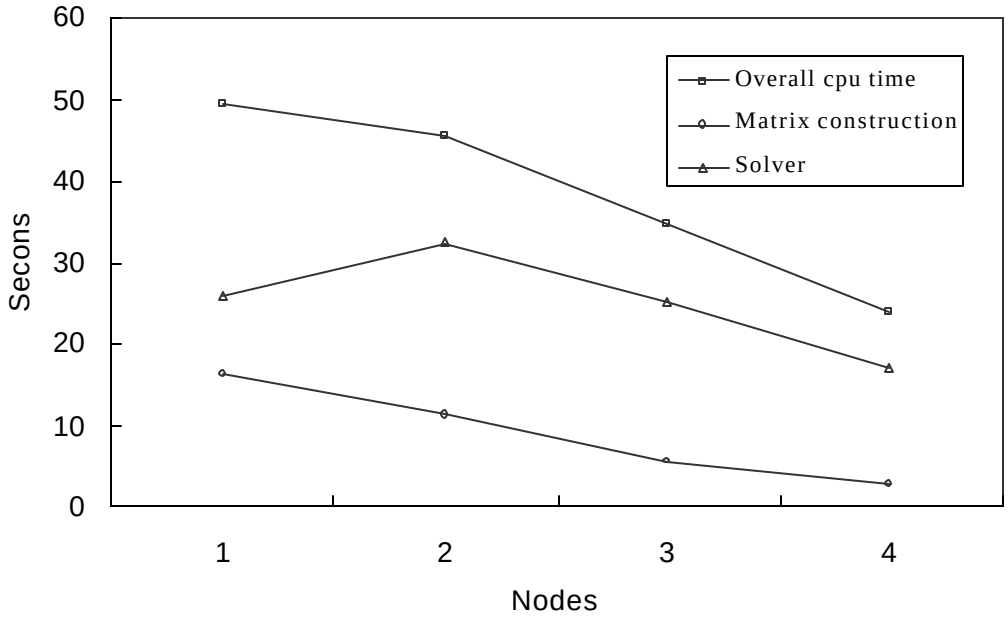


Figure 2.12 Speedup of LSOR on distributed memory machine, IBM SP2: (36,80,20) grids

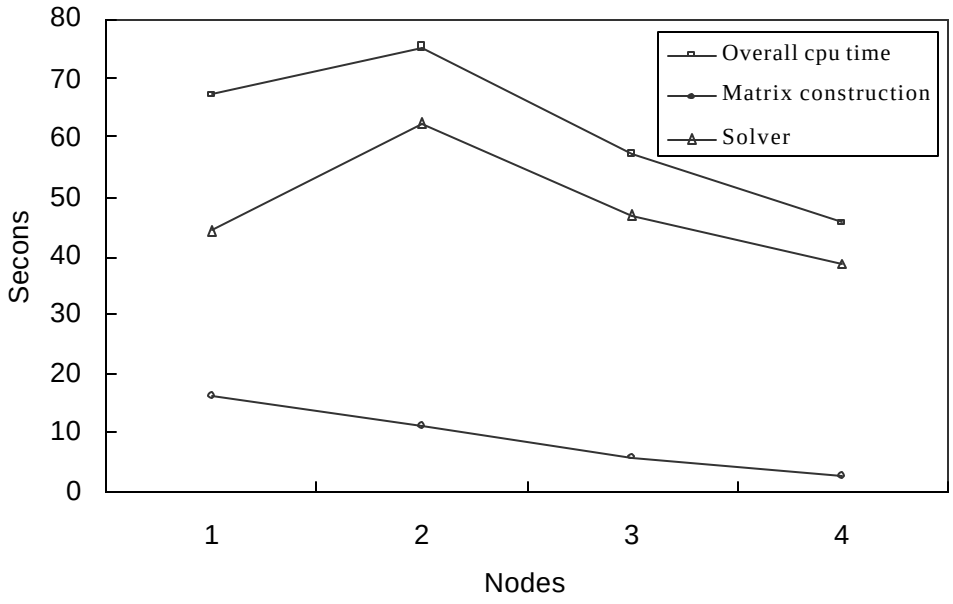


Figure 2.13 Speedup of GMRES(3) on distributed memory machine, IBM SP2: (36,80,20) grids

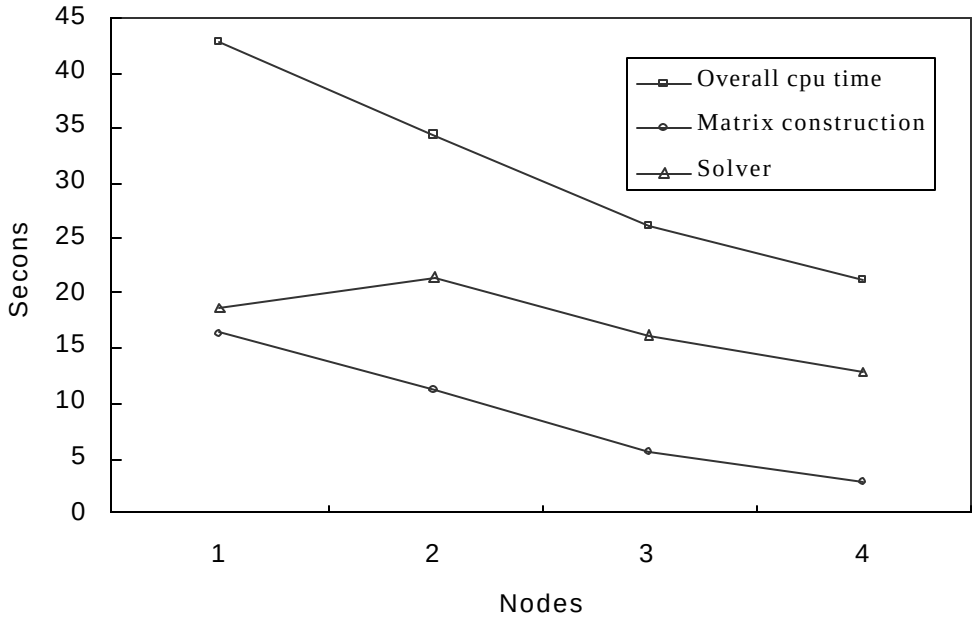


Figure 2.14 Speedup of TFQMR on distributed memory machine, IBM SP2: (36,80,20) grids

Table 2.4 Comparison of CPU time of different domain decomposition schemes on SGI Power Challenge.

Methods of solver	Nodes (decomposition)	Local grid size	CPU time (sec)	Speedup
LSOR	1	(48,48,4)	274.8	1.0
	4 (1-D, x direction)	(12,48,4)	100.4	2.7
	8 (1-D, x direction)	(6,48,4)	66.7	4.1
	4 (2-D, x and y direction)	(48,28,2)	100.8	2.7
	8 (3-D)	(24,24,2)	68.3	4.0
TFQMR	4 (1-D, x direction)	(12,48,4)	94.8	2.9
	8 (1-D, x direction)	(6,48,4)	57.5	4.8
	4 (2-D, x and y direction)	(48,28,2)	100.6	2.7
	8 (3-D)	(24,24,2)	76.0	3.6

2.6 Conclusions

Using the finite difference IMPES scheme, a new parallel black-oil simulator was implemented. The following conclusions were obtained from the study, which focused on the comparative performances of several parallel linear solvers.

1. The simple LSOR method for parallel computers was shown to be a viable tool for parallelization of reservoir simulators. The performance of this scheme is comparable to the powerful TFQMR method.
2. The performance of the parallel linear solvers on the shared and the distributed memory machines was comparable, which demonstrated the portability of MPI in parallel implementation of oil reservoir simulation programs.
3. Single-dimension domain decomposition saves communication overhead and is thus better than multidimensional divisions of the reservoir.

CHAPTER 3

DISCRETE FRACTURE MODEL FOR FRACTURED OIL RESERVOIR SYSTEM

3.1 Introduction

Computer simulation of multiphase flow through fractured porous media continues to be challenging. Important applications are in the areas of contaminant transport in porous media and in oil, gas and water flow in petroleum reservoirs. It has been recognized that most subsurface systems are fractured to a certain degree. New system characterization techniques, particularly in oil and gas reservoir engineering are making it possible to map out fracture networks, at least to a limited extent. Four distinct approaches have been employed for the simulation of fluid flow in fractured porous media; explicit discrete fractures, discrete-fracture networks, single continuum and dual continua (Bear, 1993). Historical development of the models both in the oil industry and in the hydrogeological community has been discussed in detail by Pinder, et al, (1993). Kazemi and Gilman (1993) provide a more comprehensive mathematical review of the models developed for the multiphase simulation in oil, gas and coalbed reservoirs. Pioneering efforts of Kazemi and Gilman, particularly in the development of dual-porosity models, has made possible simulation of fractured systems at the reservoir scale.

Even though this technique has served the practical purpose of simulating real reservoirs, it is recognized that this dual-continuum concept where orthogonal grid blocks are typically used to capture the thin fracture networks (Figure 3.1) is unable to represent fractures in a spatially accurate manner. It may be possible to simulate these fractured systems using single porosity models with fine grids around fractures. However, that would require an enormous number of grid blocks. Another possible approach is homogenization method using upscaling techniques to generate equivalent properties of porous medium. Unfortunately, it has generally become known that these conventional technologies average or smear out the effects of fractures in simulation due to the information lost in the upscaling process to reduce the fine grid blocks to a reasonable number. Discrete fracture models are an alternative to the single and dual porosity models. These are more common in groundwater and contaminant transport; however, few attempts at multiphase flow modeling have been made because of the computational effort required.

The model developed in this chapter is essentially to address incorporation of fractures in a spatially explicit fashion in a given domain. The model has been developed to provide an infrastructure to utilize sophisticated fracture identification and mapping methods. The complex geometry of a complex fracture network has typically required a finite element discretization of the spatial domain. Finite-element reservoir simulation models are not very common. The model developed here is conceptually similar to the model developed by Dalen (1979) for non-fractured reservoirs and hydrogeologic multiphase flow models proposed by Kaluarachchi and Parker (1989). The discrete-fracture implementation is similar, in principle to the formulation described by Huyakorn

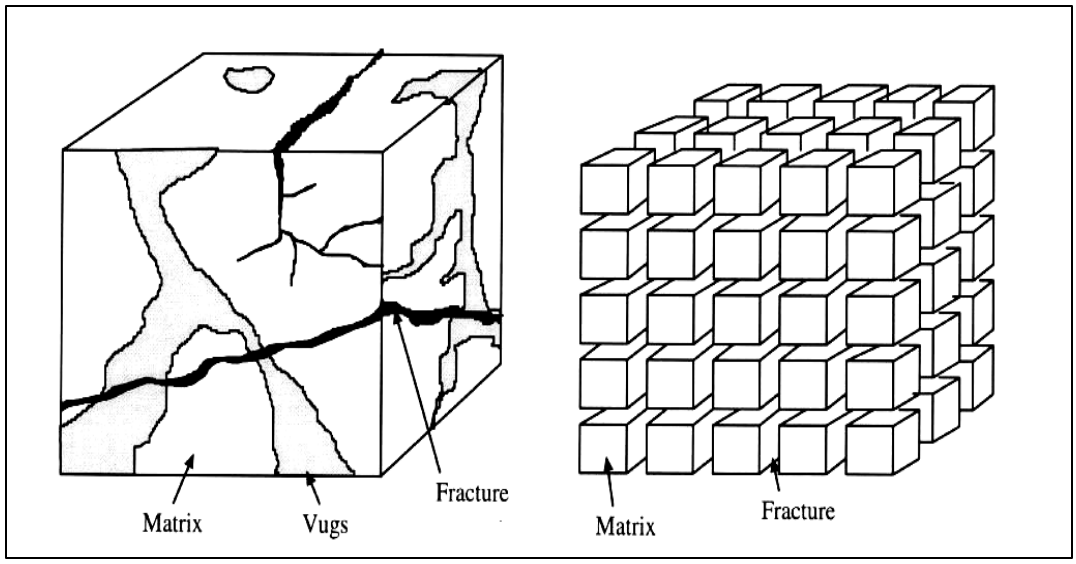


Figure 3.1 Idealization of fractured reservoir in dual porosity model.

et al. (1983) who implemented the model for a single vertical fracture. Huyakorn et al. (1994) described a three-dimensional, multiphase flow model for flow in fractured media. Slough, et al. (1999) referred to a discrete-fracture model when discussing the importance of rock matrix entry pressure on dense nonaqueous phase liquid migration in fractured geologic materials. Details of the model and implementation were not provided. Development and implementation of a discrete-fracture model to practical multiphase flow problems has not been addressed previously and is the subject of this chapter.

A two-dimensional problem is solved; fractures are represented as one-dimensional line elements while homogeneous matrix space of the system is represented by two-dimensional triangular elements. The model requires the solution of fully coupled two-phase flow of slightly compressible, immiscible fluids in the matrix and fracture elements. The topics addressed in the following sections include

- the mathematical description of the two-dimensional and two-phase flow discrete fracture model (oil and water phases)
- the development of finite element numerical algorithm for solving the governing flow equations
- results and discussions of a number of simulations performed to illustrate specific aspects of the naturally fracture system.

3.2 Development of the discrete-fracture model

3.2.1 Governing equations

In the discrete-fracture approach, multiphase flow equations are written for the porous matrix and for the fractures separately. Consider the flow of oil and water, the two immiscible phases either through matrix blocks or through fractures. The governing flow equations are derived by combining Darcy's law with equation of continuity.

$$\text{Oil phase equation: } \nabla \cdot \left(T_o \nabla \left(P_o - g_o Z\right)\right) = \frac{\partial}{\partial t}\left(f \frac{(1 - S_w)}{B_o} \right) \pm \frac{q_o}{r_{osc}} \tag{3.1}$$

$$\text{Water phase equation: } \nabla \cdot \left(T_w \left(\nabla P_o - P_c' \nabla S_w - g_w Z\right)\right) = \frac{\partial}{\partial t}\left(f \frac{S_w}{B_w} \right) \pm \frac{q_w}{r_{wsc}} \tag{3.2}$$

In equations 3.1 and 3.2, the difference between phase pressures is given by the capillary pressure, which in general, is a nonlinear function of fluid saturation.

$$P_c = P_0 - P_w = f(S_w) \tag{3.3}$$

$$S_o + S_w = 1 \tag{3.4}$$

The transmissibilities in equations 3.1 and 3.2 are given by the following expressions.

$$T_o = \frac{k_{ro} k_{absolute}}{m_o B_o} \tag{3.5}$$

$$T_w = \frac{k_{rw} k_{absolute}}{m_w B_w} \tag{3.6}$$

In the above equations, the relative permeabilities are also highly nonlinear functions of saturations. The main governing equations (3.1 and 3.2) can be formulated in terms of different sets of dependent variables (Aziz and Settari, 1979). When oil pressure and water saturation are chosen as dependent variables, equations 3.1-3.3 can be combined into the following matrix equation.

$$\begin{bmatrix} 0 & \nabla \cdot T_o \\ \nabla \cdot T_w P_c' & \nabla \cdot T_w \end{bmatrix} \times \begin{bmatrix} \nabla S_w \\ \nabla P_o \end{bmatrix} + \begin{bmatrix} \nabla \cdot T_o & 0 \\ 0 & \nabla \cdot T_w \end{bmatrix} \times \begin{bmatrix} -g_o \nabla Z \\ -g_w \nabla Z \end{bmatrix} = \begin{bmatrix} do1 & do2 \\ dw1 & dw2 \end{bmatrix} \times \begin{bmatrix} \frac{\partial S_w}{\partial t} \\ \frac{\partial P_o}{\partial t} \end{bmatrix} + \begin{bmatrix} \frac{q_o}{r_{osc}} \\ \frac{q_w}{r_{wsc}} \end{bmatrix} \tag{3.7}$$

The coefficients on the right hand side of equation 3.7 (also known as the storage matrix) can be expressed in terms of the primary variables as follows.

$$do1 = -f \frac{1}{B_o} \tag{3.8a}$$

$$do2 = f(1 - S_w)\frac{\partial(\frac{1}{B_o})}{\partial P_o} \tag{3.8b}$$

$$dw1 = f\frac{1}{B_w} \tag{3.8c}$$

$$dw2 = fS_w\frac{\partial(\frac{1}{B_w})}{\partial P_o} \tag{3.8d}$$

In the discrete-fracture model, identical equations are written for multiphase transport through fractures.

Oil phase equation for flow through fractures:

$$\nabla^* \cdot \left(T_{of}\nabla^*\left(P_{of} - g_o Z\right)\right) = \frac{\partial}{\partial t}\left(f_f\frac{(1 - S_{wf})}{B_o}\right) \pm \frac{q_{of}}{r_{osc}} \tag{3.9}$$

Water phase equation for flow through fractures:

$$\nabla^* \cdot \left(T_{wf}\left(\nabla^* P_{of} - P_{cf}^{'}\nabla^* S_{wf} - g_w Z\right)\right) = \frac{\partial}{\partial t}\left(f_f\frac{S_{wf}}{B_w}\right) \pm \frac{q_{wf}}{r_{wsc}} \tag{3.10}$$

The fracture equations are implemented using a local coordinate system, the relation of which to the global coordinate system is shown in Figure 3.2. Sets of initial conditions and boundary conditions complete the equation system.

Initial Conditions: $P_o(x, y, 0) = P_o(x, y)$ *at t=0* (3.11a)

$S_w(x, y, 0) = S_w(x, y)$ *at t=0* (3.11b)

No flow conditions (Neumann boundary conditions) are imposed at the boundary.

$$\frac{\partial P_o}{\partial x} = 0, \qquad \frac{\partial S_w}{\partial x} = 0 \;\; at\, x = 0 \, and \, x = x_0 \tag{3.12a}$$

$$\frac{\partial P_o}{\partial y} = 0, \qquad \frac{\partial S_w}{\partial y} = 0 \;\; at\, y = 0 \, and \, y = y_0 \tag{3.12b}$$

3.2.2 Finite element discretization

As mentioned previously, finite element formulation was used because of the complex system geometry. The finite element discretization of the matrix and fracture flow equations is expressed in terms of nodal oil pressures and water saturations. Standard Galerkin method was employed, wherein, the matrix was represented by linear triangular elements and the fractures by line elements. The shape functions representing the primary dependent variables are described by the following equations.

$$\Phi = a + bx + cy \tag{3.13}$$

$$a = \frac{1}{2\,Area}\left[\left(x_2 y_3 - x_3 y_2\right)\Phi_1 + \left(x_3 y_1 - x_1 y_3\right)\Phi_2 + \left(x_1 y_2 - x_2 y_1\right)\Phi_3\right] \tag{3.14a}$$

$$b = \frac{1}{2\,Area}\left[\left(y_2 - y_3\right)\Phi_1 + \left(y_3 - y_1\right)\Phi_2 + \left(y_1 - y_2\right)\Phi_3\right] \tag{3.14b}$$

$$c = \frac{1}{2\,Area}\left[\left(x_3 - x_2\right)\Phi_1 + \left(x_1 - x_3\right)\Phi_2 + \left(x_2 - x_1\right)\Phi_3\right] \tag{3.14c}$$

The shape functions for the fracture line elements are similarly given by the following expressions.

$$\Phi_1 = \frac{x - x_1^*}{length} \tag{3.15a}$$

$$\Phi_2 = \frac{x_2^* - x}{legth} \tag{3.15b}$$

x^* indicates that the calculations are in the local coordinate system (Figure 2.1). The primary dependent variables are calculated using the basis functions.

$$P_o(x, y, t) \approx \sum_{\Omega} P_o(t)\Phi(x, y)$$
$$S_w(x, y, t) \approx \sum_{\Omega} S_w(t)\Phi(x, y) \tag{3.16}$$

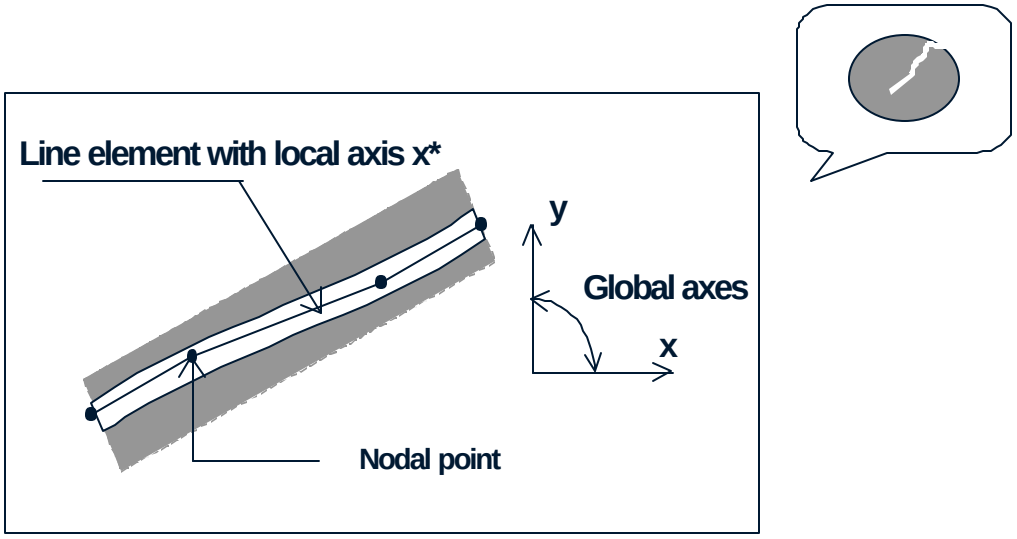


Figure 3.2 Representation of a fracture using linear line elements in a local coordinate system.

The formulation was designed to accommodate spatially variable properties since most realistic subsurface flow applications involve heterogeneous systems. The variable coefficients of storage matrix are approximated by the following functional representations.

$$\overline{do}_{1,2}(x,y,t) \approx \sum_{\Omega} do_{1,2}(t)\Phi(x,y)$$
$$\overline{dw}_{1,2}(x,y,t) \approx \sum_{\Omega} dw_{1,2}(t)\Phi(x,y) \qquad (3.17)$$

The non-constant transmissibility terms in the flow equations are usually subjected to upstream mobility weighting (Dalen, 1979; Huyakorn, 1983; Kaluarachchi and Parker, 1989). In the upstream weighting formulation, the generalized flux between a pair of nodes is given by

$$(q_l)_{i,j} = \tilde{T}_l \nabla \cdot (\nabla P_l - g_l \nabla Z)_{i,j} \qquad (3.18)$$

where the upstream weighted transmissibility is

$$\tilde{T}_l = \begin{cases} T_{l,i} & \text{if } P_{l,i} - g_{l,i} Z_i \geq P_{l,j} - g_{l,j} Z_j \\ T_{l,j} & \text{if } P_{l,j} - g_{l,j} Z_j \geq P_{l,i} - g_{l,i} Z_i \end{cases} \qquad (3.19)$$

The discretized-integral forms of equations 3.1 and 3.2 are obtained by applying Green's theorem and the Galerkin formulation.

$$-\sum_{\Omega}^{N}\int[(\tilde{T}_o\nabla\Phi\cdot\nabla\Phi)P_o]d\Omega+\sum_{S}^{S}\int(\tilde{T}_o\nabla P_o\Phi\cdot n)d\Omega+\sum_{\Omega}^{N}\int[(\tilde{T}_o\bar{g}_o\nabla\Phi\cdot\nabla\Phi)Z]d\Omega$$

$$-\sum_{S}^{S}\int(\tilde{T}_o\bar{g}_o\nabla Z\Phi\cdot n)d\Omega=\sum_{\Omega}^{N}\int[(\overline{do}_1\Phi\cdot\Phi)\frac{\partial S_w}{\partial t}]d\Omega+\sum_{\Omega}^{N}\int[(\overline{do}_2\Phi\cdot\Phi)\frac{\partial P_o}{\partial t}]d\Omega+ \quad (3.20)$$

$$\sum_{\Omega}^{N}\int(\frac{q_o}{r_{osc}}\Phi)d\Omega \quad \text{----------oil phase}$$

$$-\sum_{\Omega}^{N}\int[(\tilde{T}_w\nabla\Phi\cdot\nabla\Phi)P_o]d\Omega+\sum_{S}^{S}\int(\tilde{T}_w\nabla P_o\Phi\cdot n)d\Omega+\sum_{\Omega}^{N}\int[(\overline{T}_w\tilde{P}_c'\nabla\Phi\cdot\nabla\Phi)S_w]d\Omega$$

$$-\sum_{S}^{S}\int(\tilde{T}_w\overline{P}_c'\nabla S_w\Phi\cdot n)d\Omega+\sum_{\Omega}^{N}\int[(\tilde{T}_w\bar{g}_w\nabla\Phi\cdot\nabla\Phi)Z]d\Omega-\sum_{S}^{S}\int(\tilde{T}_w\bar{g}_w\nabla Z\Phi\cdot n)d\Omega \quad (3.21)$$

$$=\sum_{\Omega}^{N}\int[(\overline{dw}_1\Phi\cdot\Phi)\frac{\partial S_w}{\partial t}]d\Omega+\sum_{\Omega}^{N}\int[(\overline{dw}_2\Phi\cdot\Phi)\frac{\partial P_o}{\partial t}]d\Omega+\sum_{\Omega}^{N}\int\frac{q_w}{r_{wsc}}\Phi d\Omega \quad \text{---water phase}$$

The Neumann boundary conditions in equation 3.12, $\tilde{N}P_o\backslash n = 0$ and $\tilde{N}S_w\backslash n = 0$, allowed cancellation of the boundary flux terms. The integral equations are written in matrix form as follows:

$$\begin{bmatrix}0 & Ao\\ Aw_{pc} & Aw\end{bmatrix}\times\begin{bmatrix}S_w\\ P_o\end{bmatrix}+\begin{bmatrix}Bo1 & Bo2\\ Bw1 & Bw2\end{bmatrix}\times\begin{bmatrix}\frac{\partial S_w}{\partial t}\\ \frac{\partial P_o}{\partial t}\end{bmatrix}=\begin{bmatrix}Q_o\\ Q_w\end{bmatrix}+\begin{bmatrix}Go\\ Gw\end{bmatrix} \quad (3.22)$$

$$Ao=\sum_{\Omega}^{N}\int(\tilde{T}_o\nabla\Phi\cdot\nabla\Phi)d\Omega \quad (3.23a)$$

$$Aw=\sum_{\Omega}^{N}\int(\tilde{T}_w\nabla\Phi\cdot\nabla\Phi)d\Omega \quad (3.23b)$$

$$Aw_{pc} = -\sum_{\Omega}^{N} \int (\tilde{T}_w \overline{P}_c' \nabla\Phi \cdot \nabla\Phi) d\Omega \tag{3.23c}$$

$$Bo1 = \sum_{\Omega}^{N} \int (\overline{do}1\Phi \cdot \Phi) d\Omega \tag{3.23d}$$

$$Bo2 = \sum_{\Omega}^{N} \int (\overline{do}2\Phi \cdot \Phi) d\Omega \tag{3.23e}$$

$$Bw1 = \sum_{\Omega}^{N} \int (\overline{dw}1\Phi \cdot \Phi) d\Omega \tag{3.23f}$$

$$Bw2 = \sum_{\Omega}^{N} \int (\overline{dw}2\Phi \cdot \Phi) d\Omega \tag{3.23g}$$

$$Q_l = \sum_{\Omega}^{N} \int (\frac{q_l}{r_{lsc}} \Phi) d\Omega \tag{3.23h}$$

$$G_l = \sum_{\Omega}^{N} \int [(\tilde{T}_l \bar{g}_l \nabla\Phi \cdot \nabla\Phi) Z] d\Omega - \sum_{S}^{S} \int (\tilde{T}_l \bar{g}_l \nabla Z \cdot n) d\Omega \tag{3.23i}$$

Numerical oscillations are possible when solving multiphase flow problems using the pressure-saturation dependent variable set (Dalen, 1979). To overcome this difficulty and to preserve the computational simplicity of the conventional Galerkin method, a lumping procedure was used (Dalen, 1979 and Kaluarachchi and Parker, 1989). In the lumping scheme employed, the storage matrix is diagonalized as follows:

$$\begin{bmatrix} Bo1_{(1,1)} + Bo1_{(1,2)} + Bo1_{(1,3)} & 0 & 0 \\ 0 & Bo1_{(2,1)} + Bo1_{(2,2)} + Bo1_{(2,3)} & 0 \\ 0 & 0 & Bo1_{(3,1)} + Bo1_{(3,2)} + Bo1_{(3,3)} \end{bmatrix} \tag{3.24}$$

Elemental and global matrices were constructed using standard integration procedures. In this application, two separate sets were generated, one for the matrix system and the other for the fracture network. The following general closed form integration formulae were applied.

$$\text{for linear line element : } \int_{\Omega}(\Phi_1^l\Phi_2^m)d\Omega = \frac{l!\,m!}{(l+m+1)!}\,Length_{\Omega} \tag{3.25a}$$

$$\text{for linear triangular element : } \int_{\Omega}(\Phi_1^l\Phi_2^m\Phi_3^n)d\Omega = \frac{l!\,m!\,n!}{(l+m+n+1)!}\,2\,Area_{\Omega} \tag{3.25b}$$

When upstream weighting and functional representation as described previously are applied, the variable coefficients can be taken out of the integral equations and shape functions can be integrated apriori. Using the coefficients of shape functions (equation 3.14), relevant equations for the problem at hand are given below.

$$\text{for line element : } \begin{cases} \displaystyle\int_{\Omega}(\nabla\Phi\cdot\nabla\Phi)d\Omega = \frac{1}{Length_{\Omega}}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \\ \displaystyle\int_{\Omega}(\Phi\cdot\Phi)d\Omega = \frac{Length_{\Omega}}{6}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \end{cases} \tag{3.26a}$$

$$\text{for triangular element : } \begin{cases} \displaystyle\int_{\Omega}(\nabla\Phi\cdot\nabla\Phi)d\Omega = \frac{1}{4Area_{\Omega}}\begin{bmatrix} b_1b_1+c_1c_1 & b_1b_2+c_1c_2 & b_1b_3+c_1c_3 \\ b_2b_1+c_2c_1 & b_2b_2+c_2c_2 & b_2b_3+c_2c_3 \\ b_3b_1+c_3c_3 & b_3b_2+c_3c_2 & b_3b_3+c_3c_3 \end{bmatrix} \\ \displaystyle\int_{\Omega}(\Phi\cdot\Phi)d\Omega = \frac{Area_{\Omega}}{12}\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \end{cases} \tag{3.26b}$$

The two global matrices are superposed as shown in Figure 3.3 and the combined problem solved.

3.3 Numerical results

3.3.1 Model problem

The applicability of using the above scheme was tested on a two-dimensional test problem, which was created using the FracMan^{TM} software of Golder Associates, Inc. Details of the geologic data input and the methodology of fracture network generation are discussed in detail in Forster, et al. (1998). Detailed initial fracture map generated using field data is shown in Figure 3.4. In creation of the discrete-fracture networks, fracture

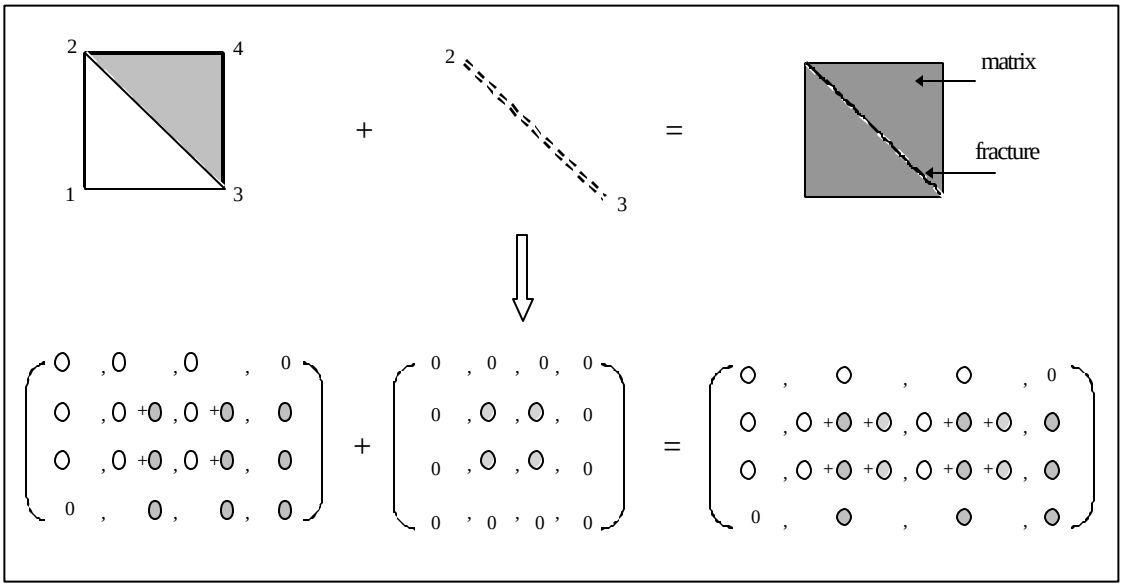


Figure 3.3 Superposition of the global matrices of the matrix and the fracture components of the discrete fracture domain.

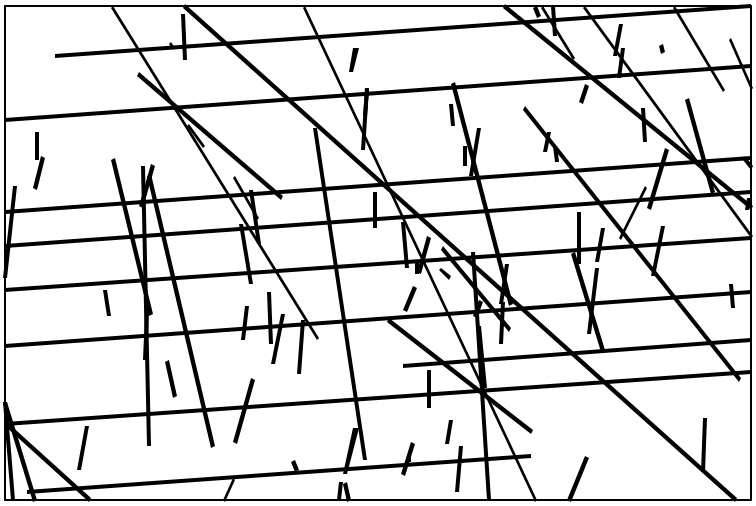


Figure 3.4 The original fracture map generated using field outcrop data.

properties such as mean orientation and dispersion of orientation, mean fracture density and spacing, fracture-size statistics, and connectivity, collected from outcrop measurements were used. For the test problem, about 10 % of the fracture lines having dominant length were chosen from the detailed initial fracture image. The fracture network for the flow simulations is shown in Figure 3.5. There were a total of 193 nodes, 341 triangular elements and 139 fracture line elements in the rectangular domain of 18.3 m by 18.3 m. Mesh generation was accomplished using the Voronoi algorithm (Fortune, 1987). The finite element grid is also shown in Figure 3.5. A water flood was simulated with an injection well in the north-east corner and a production well at the south-west corner, diagonally opposite. The relative permeability to oil and water were symmetrical. Relative permeabilities and other relevant properties are tabulated in Table 3.1. The total fluid injection and production rates were identical and kept constant over the duration of one simulation run. Several different rates and permeability contrasts were simulated.

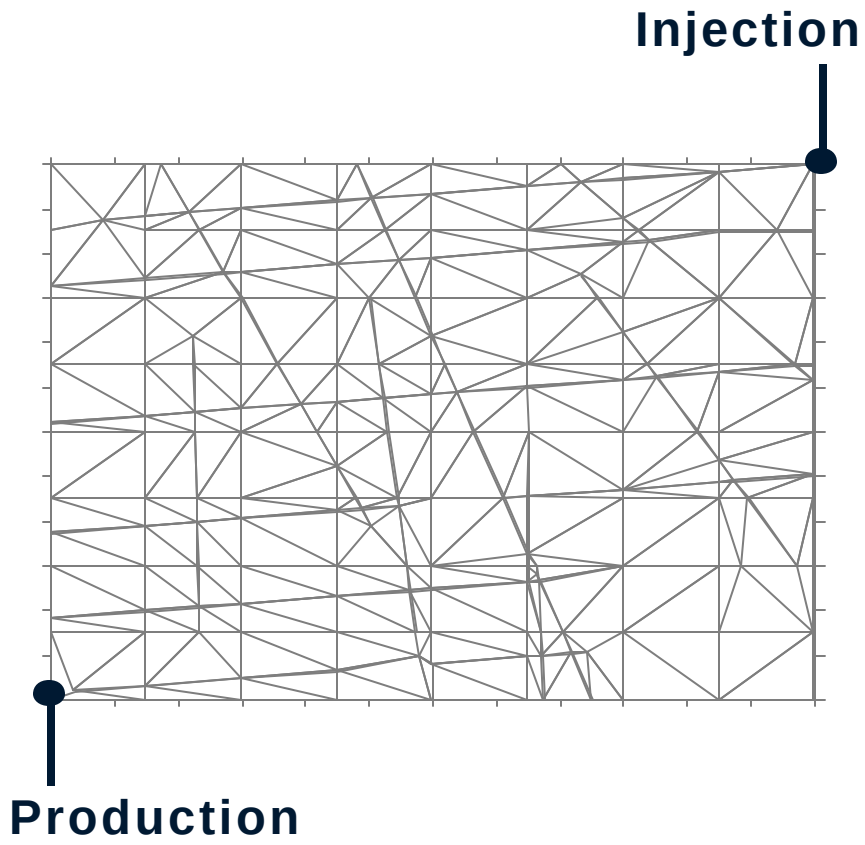


Figure 3.5 Discrete fracture test domain; also shown the finite element discretization.

Table 3.1 Input data for discrete fracture simulations.

Porosity, fraction	0.2
Initial pressure, atm	64.6
Initial oil saturation, fraction	0.8
Stock tank oil density, g/cm^3	0.85
Stock tank water density, g/cm^3	1.0
Slope of oil formation factor versus pressure, atm^{-1}	-0.0003
Slope of oil viscosity versus pressure, cp/atm^{-1}	0.0007
Water formation factor, RC volume/SC volume	1.0
Water viscosity, cp	1.0

Relative permeability and capillary pressure data :

S_w	K_{rw}	K_{ro}	P_{cow}(atm)
0.0	0.00	1.0	0.61
0.1	0.00	1.0	0.61
0.2	0.00	1.0	0.61
0.3	0.0675	0.3675	0.54
0.4	0.12	0.27	0.48
0.5	0.1875	0.1875	0.41
0.6	0.27	0.12	0.34
0.7	0.3675	0.0675	0.27
0.8	1.0	0.0	0.20

3.3.2 Validation of the model

The discrete-fracture model, so developed was validated using two methods. In the first, the finite element formulation was tested on a nonfractured domain of the size described previously and results were compared to results from a finite difference model. Identical water flood problem was simulated using the two models. The finite difference simulator used was developed by the Computer Modeling Group (CMG). It was their black-oil model, IMEX. The output from a 20 by 20 grid blocks model was compared to the results from the straightforward finite-element discretization (regular) of the domain. The finite element grid is shown in Figure 3.6. These are single-continuum, single-porosity models. The comparison of the recovery curves from the two models is shown in Figure 3.7. As can be seen from the figure, the finite element model matches the well-established finite difference model almost exactly. This shows that the finite element development, the solution method and the numerical algorithms employed are fundamentally sound.

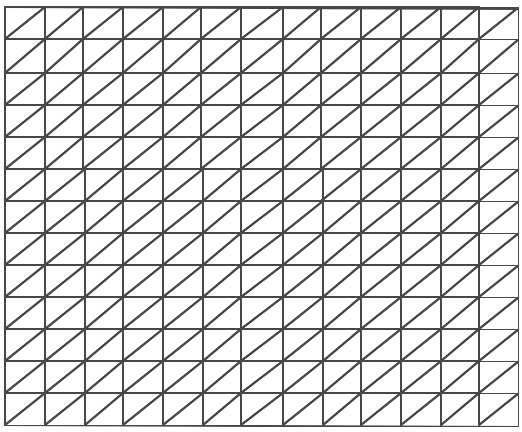


Figure 3.6 Finite element grid used for comparison with a commercial finite difference simulator.

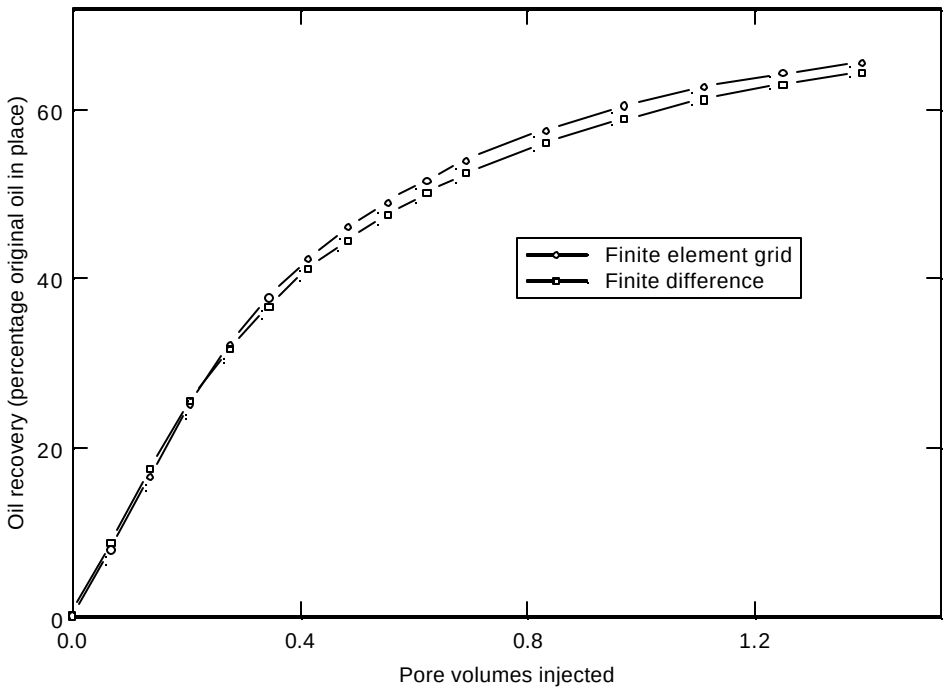
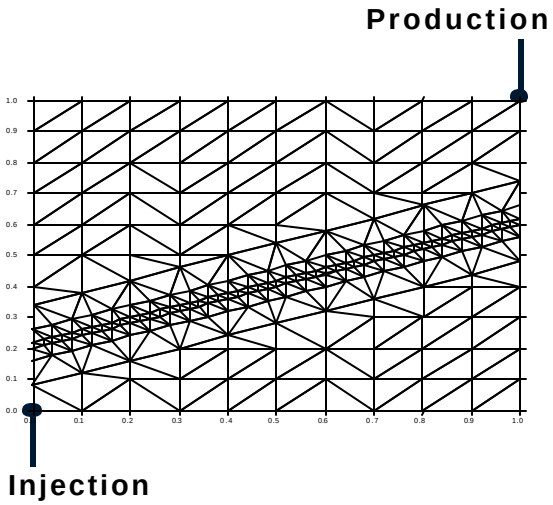
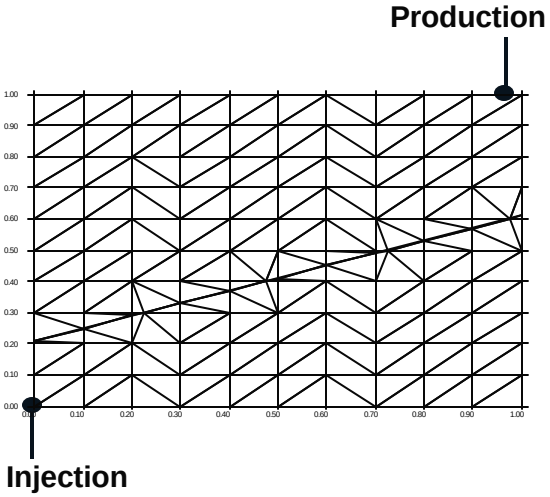


Figure 3.7 Comparison of the recovery curves for the finite element and a finite difference models of the nonfractured system.

Results from a much simpler discrete-fracture model were also compared to results from a model that allowed explicit fracture representation. The two model domains with their respective finite element discretizations are shown in Figure 3.8. There were 262 nodes and 500 triangular elements in the explicit fracture representation while the discrete-fracture model divided the domain of 0.3 m by 0.3 m into 136 nodes and 228 elements, as depicted in Figure 3.8. The explicit fracture representation is essentially a fine-grid discretization of a domain containing a fracture. It is a single-continuum, single-porosity model which was validated earlier using a finite difference model. The model parameters used in these simulations are summarized in Table 3.1. A permeability contrast of a 1000 was employed. This meant that all the fine elements representing the fracture had a permeability thousand times the permeability of the elements representing the surrounding matrix. Comparison of the oil recovery curves for the two models are shown in Figure 3.9. The two curves are reasonably close. Comparison of the water saturation contours at 0.7 pore volumes (PV) injected is presented in Figure 3.10. The two saturation maps agree reasonably well. However, the water saturations downstream are slightly higher in the discrete-fracture model. The water velocities through the one-dimensional fracture representation in the discrete-fracture model are higher compared to the explicit fracture representation. Considering the fundamental differences between the two models, the recovery curves and the saturation maps in the two simulation methods (discrete-fracture and explicit-fracture) are still reasonably close. It should be noted that even to simulate one fracture, approximately, twice the number of elements were required in the explicit-fracture representation, compared to the discrete-fracture method. For even a small number of



a)



b)

Figure 3.8 A simple, single fractured problem modeled using a) explicit fine grid representation and b) discrete fracture approach.

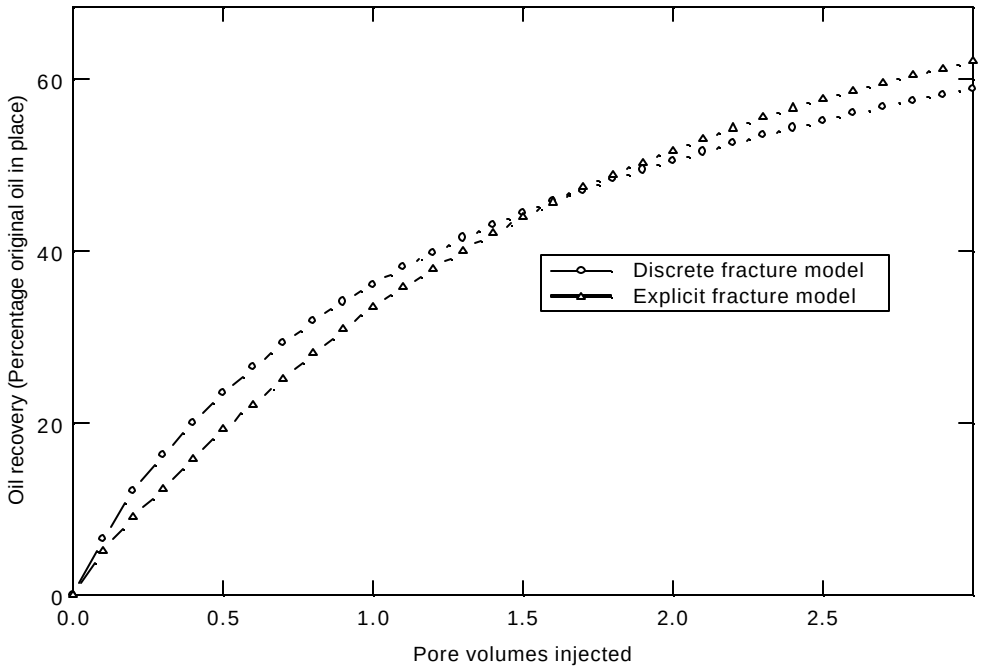
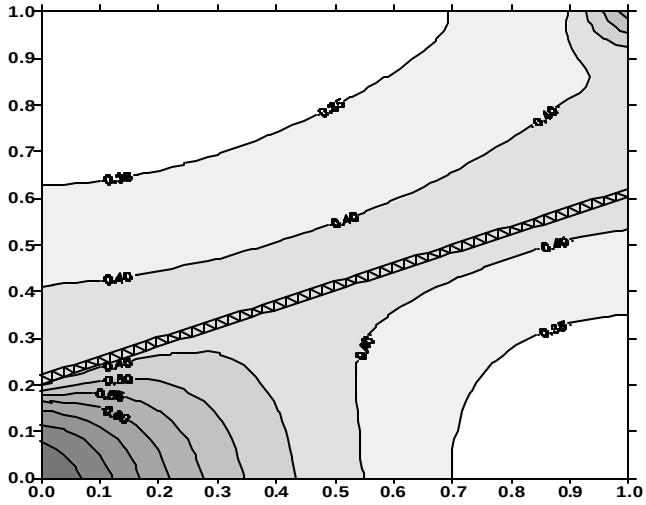
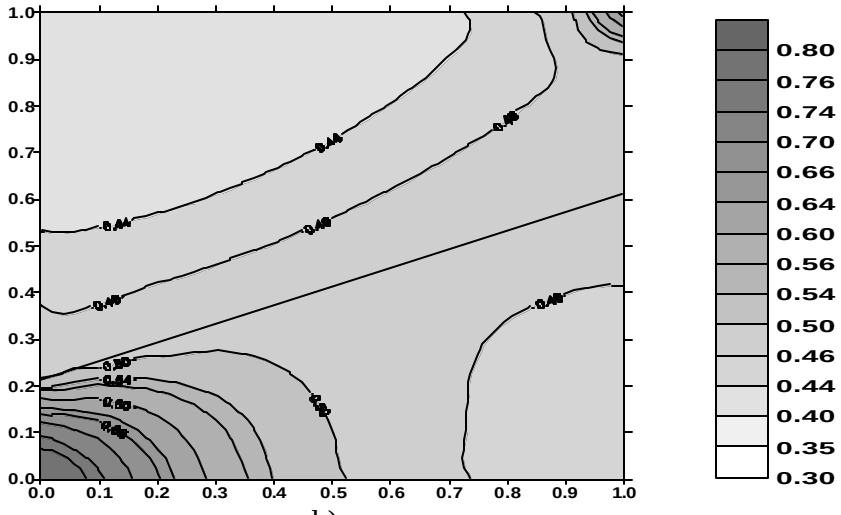


Figure 3.9 Oil recovery comparisons for the explicit fracture and the discrete fracture models.



a)



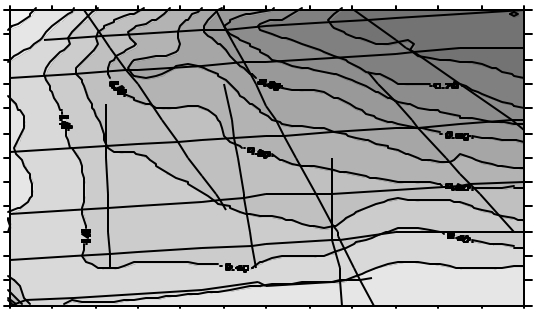
b)

Figure 3.10 Water saturation contours for a) the explicit fracture model and b) the discrete fracture model at 0.7 PV water injected. Absolute permeability was 1mD and the permeability contrast was 1000.

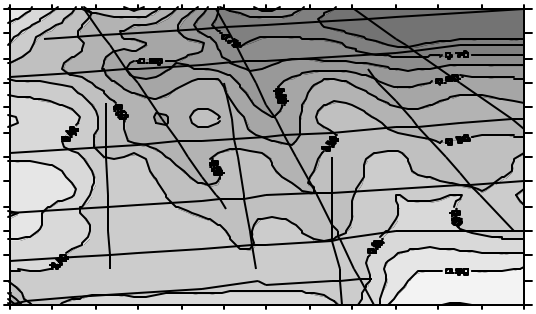
fractures, the explicit fracture method will become computationally impractical. Thus, a framework has been developed to simulate the presence of fractures in a spatially explicit manner using the discrete-fracture approach.

3.3.3 Effect of fracture to matrix permeability contrasts

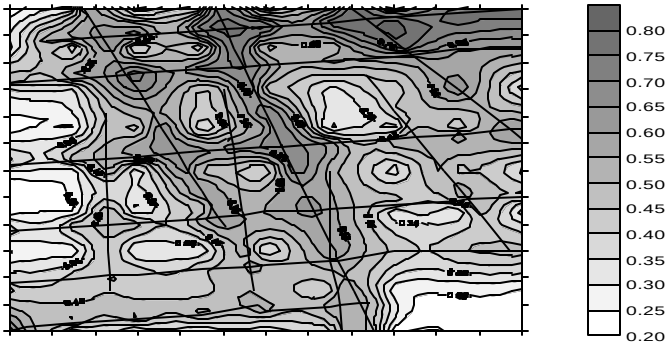
Water saturation profiles after a total of 0.7 pore volumes (PV) of water injected for each of the three permeability contrasts are shown in Figures 3.11. Permeability contrasts ($k_{fracture}$:k_{matrix}) explored in this analysis represent different ratio between fracture and matrix permeability (10:1, 100:1 and 1000:1). Other simulation parameters are given in Table 3.1. Since permeability contrast between the fractures and the matrix is small in the 10:1 case, the saturation profiles are more or less symmetrical (Figure 3.11). Figure 3.11 also shows that increasing permeability ratio to 100:1 and then to 1000:1 causes the fractures to have a greater impact on the water flood. For a permeability contrast of 1000:1, the flood path is uniquely fracture network dependent. In addition, it is apparent that increasing the permeability contrasts causes a net reduction in oil recovery for a given volume of injected water (Table 3.2). The oil recovery behavior is illustrated in Figure 3.12. As the permeability contrast increases, the water cut in the produced fluids increases more rapidly leading to more gradual oil recovery and eventually to reduced total recovery (Table 3.2). Thus, localization of flow in the higher permeability fractures causes reduced sweep of oil from the intervening matrix blocks.



a) permeability contrast – 1:10



b) permeability contrast – 1:100



c) permeability contrast – 1:1000

Figure 3.11 Effect of the matrix to fracture permeability contrast on water saturations at 0.7 PV injected; Absolute matrix permeability was 1mD.

Table 3.2 Oil recoveries (Percentage original oil in place, OOIP recovered after injecting 1.4 pore volumes of water) at different matrix to fracture permeability ratios. Absolute permeability was 1 mD in all simulations.

Model	Permeability contrast	Rate (m^3/day)	Recovery % OOIP
Discrete fracture	10 to 1	0.014	65
Discrete fracture	100 to 1	0.014	61
Discrete fracture	1000 to 1	0.014	54

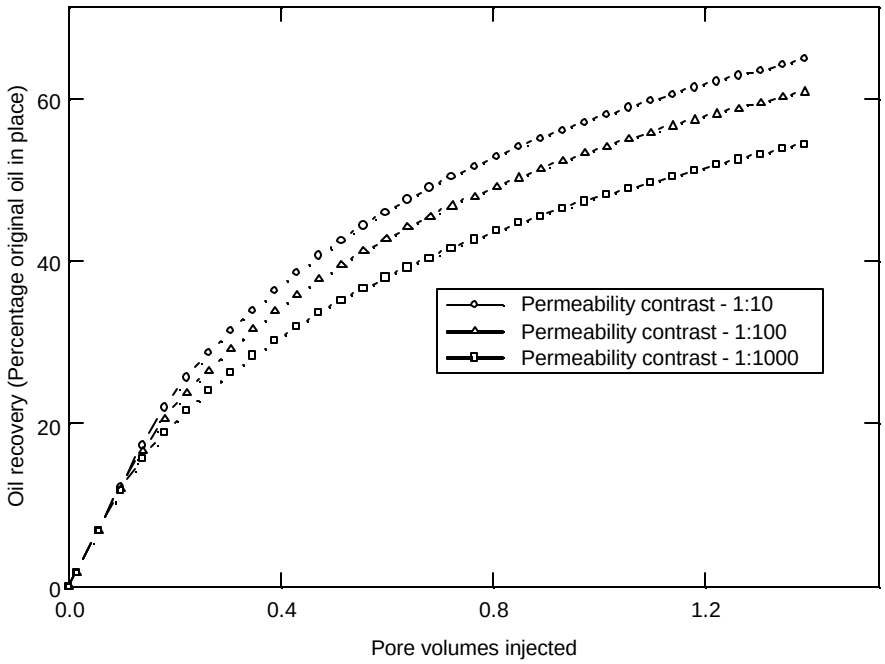


Figure 3.12 Oil recoveries at different matrix to fracture permeability contrast. Absolute matrix permeability was 1mD.

3.3.4 Effect of absolute matrix permeability

It should be noted that the absolute permeability of the matrix in all of the previous simulations was 1mD. The effect of the absolute matrix permeability on oil recovery was examined at the highest permeability contrast (1:1000). The recovery curve comparison is shown in Figure 3.13. As the absolute matrix permeability increases, as expected, the total recovery increases. Oil is also recovered at a faster rate. The water saturation contours for the two absolute permeabilities are compared in Figure 3.14. Fracture network plays a less significant role in determining oil recovery and fluid distributions. There is less oil bypassing and more even water saturation distribution at higher absolute matrix permeability.

3.3.5 Effect of injection rate

Oil recovery from fractured systems is a complex interplay of permeability contrast, absolute permeabilities and flow rates. Series of simulations were performed at various injection rates and at the high permeability contrast to assess the effect of flow rate on oil recovery and fluid distributions. The recovery curves for four different flow rates are compared in Figure 3.15. The absolute permeability in these simulations was 1 mD. The oil recoveries at 1.4 PV injected for the four flow rates are compared in Table 3.3. In general, as the flow rate increases, oil recovery decreases. The water saturation distributions for three of the flow rates are compared in Figure 3.16. It is clear from this figure that higher injection rate leads to greater oil bypassing and thus lower recovery. Similar trends were observed for simulations performed at 100 mD absolute matrix permeability.

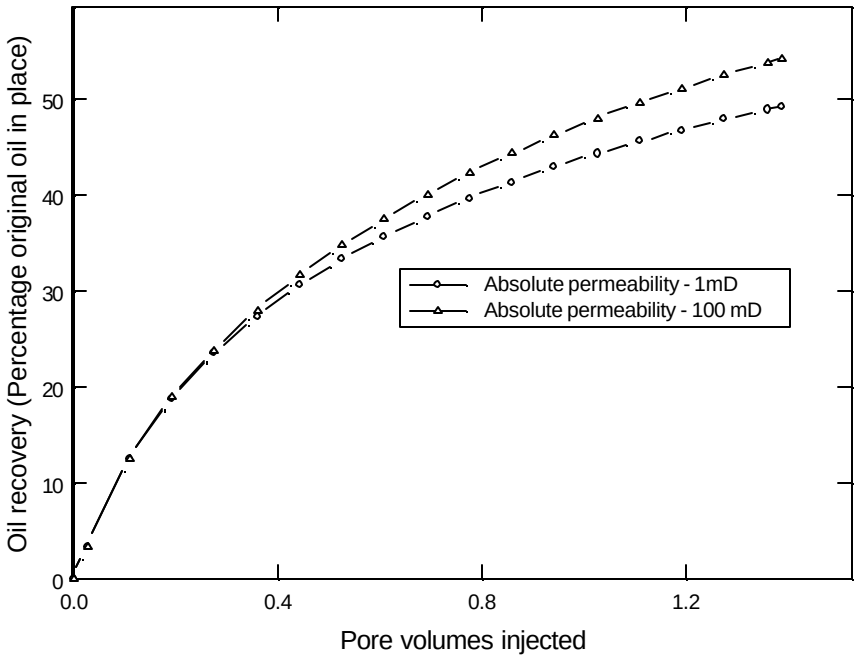
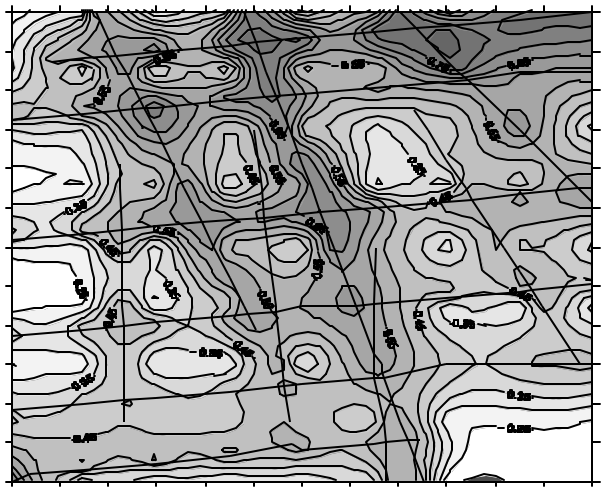
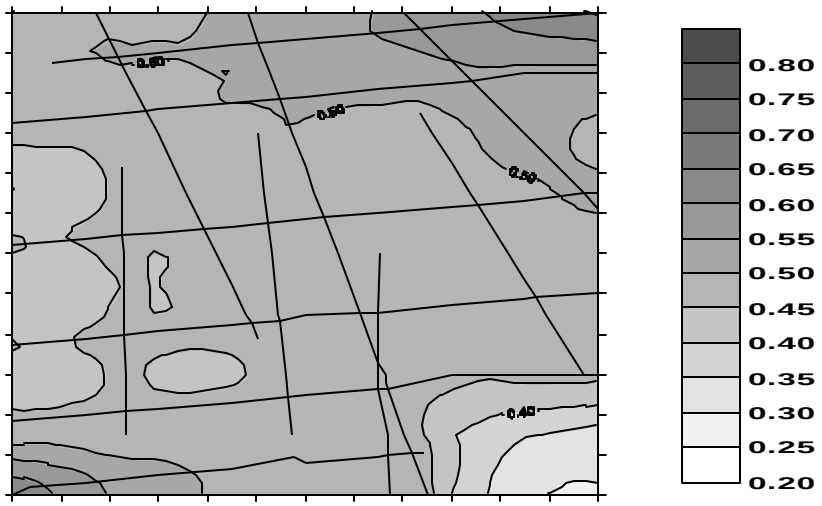


Figure 3.13 Oil recoveries at two different absolute permeabilities. Permeability contrast was 1:1000.



a)



b)

Figure 3.14 Water saturation contours at 0.7 PV water injected for two different absolute matrix permeabilities a) 1 mD and b) 100 mD. The matrix to fracture permeability contrast was 1:1000 and flow rate was 0.32 m^3/day.

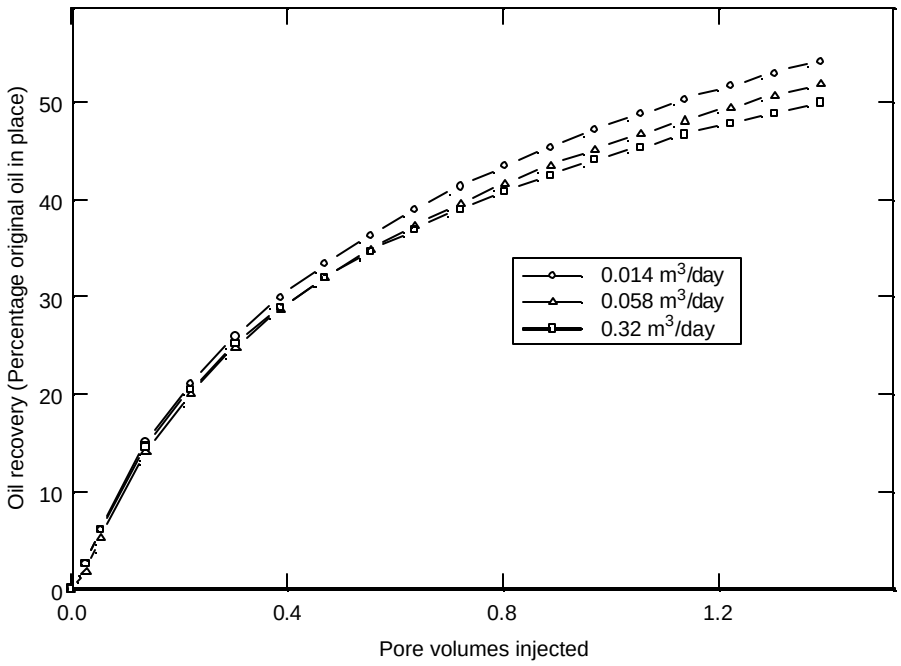
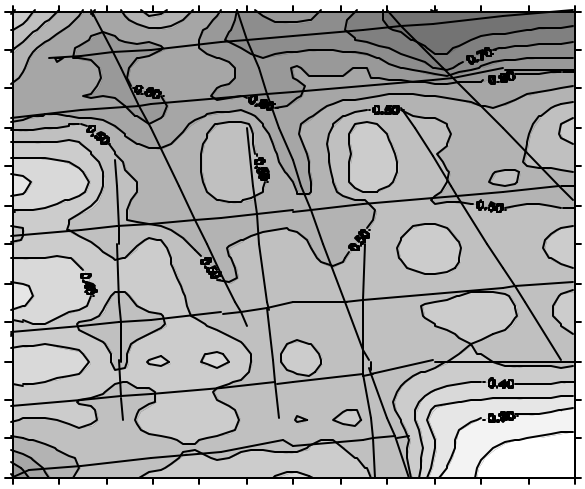


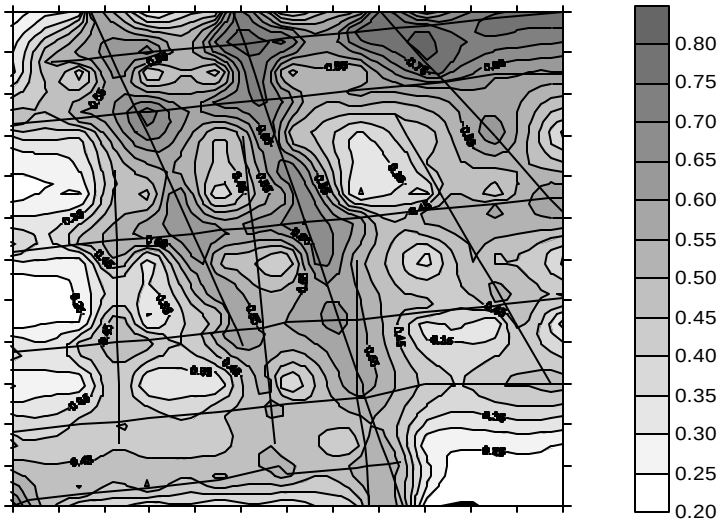
Figure 3.15 Effect of water injection rate on oil recoveries; absolute matrix permeability was 1 mD and the matrix to fracture permeability contrast was 1:1000.

Table 3.3 Oil recoveries (Percentage original oil in place, OOIP recovered after injecting 1.4 pore volumes of water) at different absolute permeabilities and injection rates; permeability contrast was 1000 to 1 in all the simulations.

Model	Absolute permeability	Rate (m^3/day)	Recovery % OOIP
Discrete fracture	1 mD	0.014	54
Discrete fracture	1 mD	0.058	52
Discrete fracture	1 mD	0.144	50
Discrete fracture	1 mD	0.320	50
Discrete fracture	100 mD	0.144	56
Discrete fracture	100 mD	0.180	55
Discrete fracture	100 mD	0.208	54
Discrete fracture	100 mD	0.320	52



a)



b)

Figure 3.16 Water saturation contours at 0.7 PV water injected at injection rates of a) 0.014 m^3/day and b) 0.32 m^3/day. The absolute matrix permeability was 1mD and matrix to fracture permeability contrast was 1:1000.

Mattax and Kyte (1962) identified “critical injection rate” at which the water advance through the fracture is too rapid for efficient transfer of oil from the matrix to the fracture. In this work, even though the oil recovery reduced with higher injection rate, it was not possible to identify a single rate at which there was a fundamental change in the oil recovery mechanism. It is considered that results of the Mattax and Kyte (1962) experimental and analytical work apply to idealized matrix/fracture systems of either one vertical or horizontal fracture system. When a complex fracture network is being simulated, the recoveries are also strongly dependent on the nature of that network. The mechanistic differences between displacements at low and high rates are captured in Figure 3.16.

3.3.6 Effect of capillary pressure

Most of the previous studies on imbibition in fractured reservoirs assume zero capillary pressure in the fractures (Terez and Firoozabadi, 1999). The discrete-fracture model developed here allows representation of different sets of capillary pressures for the matrix and for the fractures. In addition it also allows input of different sets of capillary pressures in different portions of the domain. All of the previous simulations were performed with identical capillary pressure-saturation relationships in the matrix and in the fractures (Table 3.1). The effect of capillary pressure on oil recovery was examined by reducing the capillary pressure at the fracture nodes to 10% of the base values shown in Table 3.1. Capillary pressure in the fracture appeared to have a significant influence on recovery at all injection rates only at high matrix permeability. Oil recoveries at the two sets of capillary pressures are plotted in Figure 3.17. The matrix permeability in these

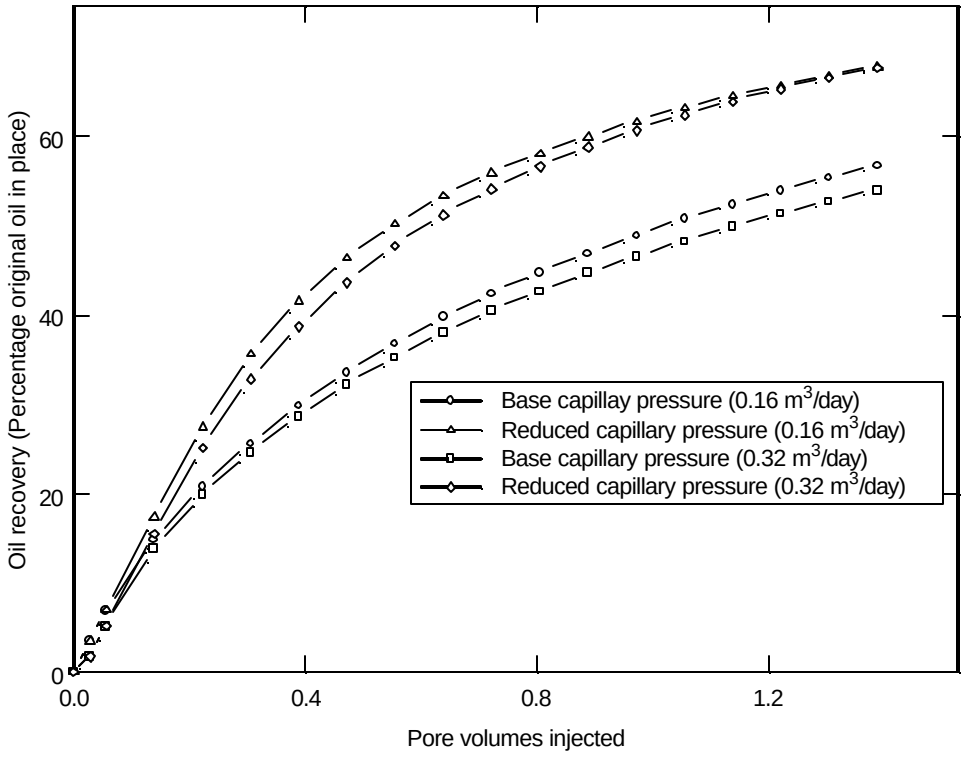


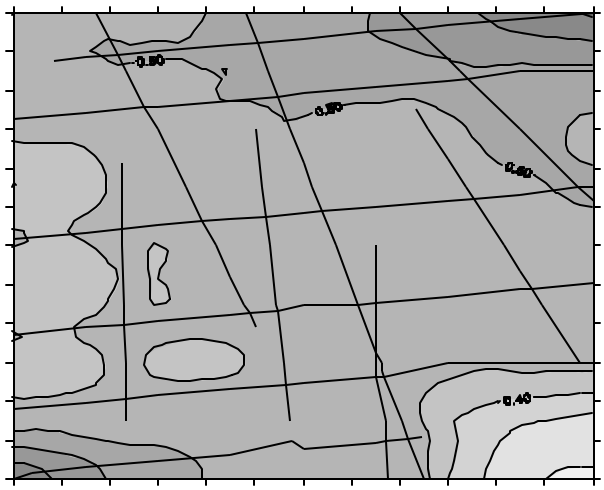
Figure 3.17 Effect of fracture capillary pressure on oil recoveries at two different injection rates. Absolute permeability was 100 mD and the fracture to matrix permeability contrast was 1:1000.

simulation was 100 mD. At the lower fracture capillary pressure, water appears to imbibe more efficiently into the matrix driving oil into the fracture network. This is true for both the low and the high flow rates. The saturation maps at the two capillary pressure sets clearly illustrate this point (Figure 3.18). When the matrix permeability is reduced to 1 mD, and at high flow rate, there is little change in oil recovery when the fracture capillary pressure is reduced. However, when the flow rate is lower, more oil is recovered at lower fracture capillary pressure (Figure 3.19). These sets of simulations indicate that oil recovery in fractured reservoirs depends on a complex interplay of capillary and viscous effects, which in turn are controlled by fracture and matrix permeabilities, fracture and matrix capillary pressures and fluid injection-production rates. The model developed here is a tool that helps identify these intricate effects by performing systematic studies.

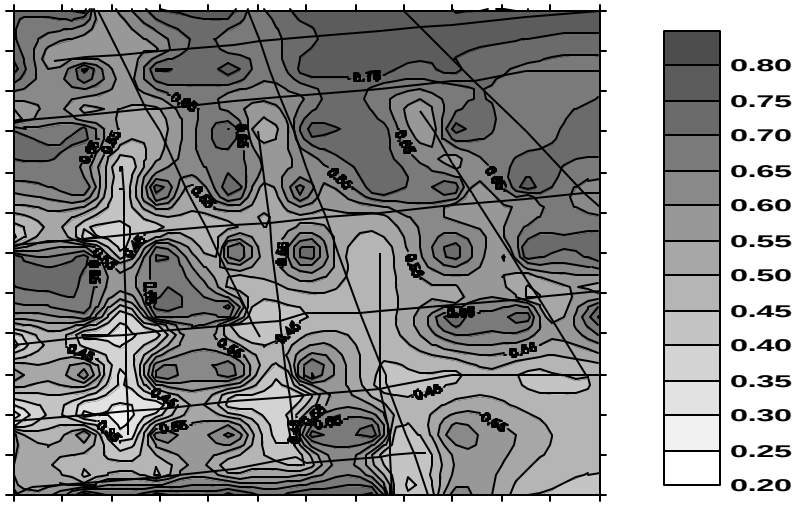
3.3.7 Comparison of the discrete fracture model with different homogenization approaches

Homogenization method developed by Koebbe (1998) was used to determine effective permeability values for the fractured domain shown in Figure 3.5; different permeability contrasts provided different effective values. Based on a rectangular grid system, effective properties are derived by finding an approximate numerical solution to the following system of elliptic partial differential equations governing flow through porous media.

$$- \nabla \cdot v^{\in} = f \qquad (3.27)$$



a)



b)

Figure 3.18 Water saturation contours at 0.7 PV injected for two different sets of fracture capillary pressures; a) base capillary pressure (Table 3.1) and b) 10 % of the base capillary pressure. Absolute permeability was 100 mD and matrix to fracture permeability contrast was 1000.

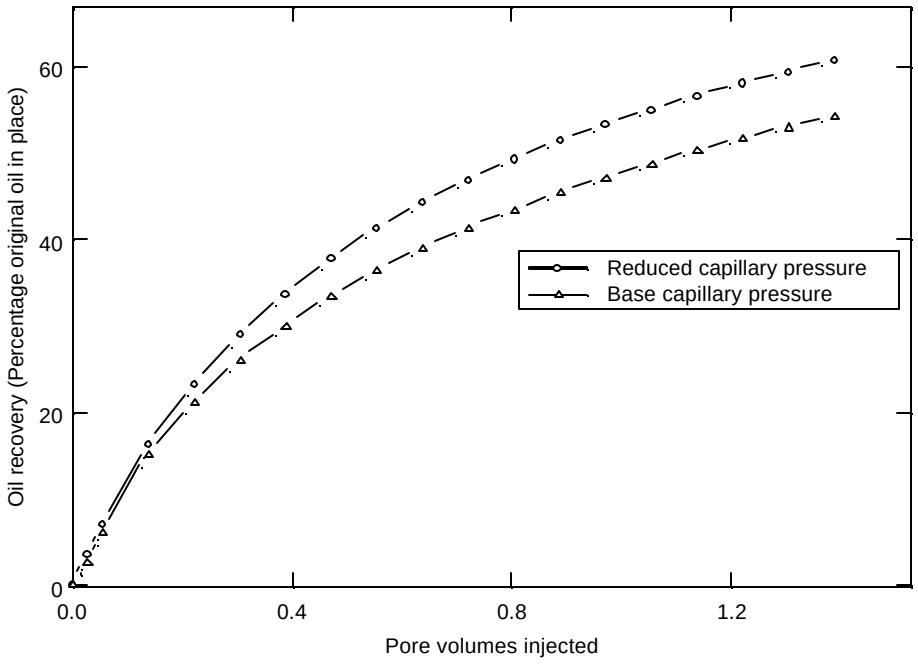


Figure 3.19 Oil recoveries at two sets of fracture capillary pressures; absolute permeability was 1 mD and the flow rate was 0.014 m^3/day. The matrix to fracture permeability contrast was 1000.

where $v^{\epsilon} = - K^{\epsilon} \nabla h^{\epsilon}$. In equation 3.27, K^{ϵ} is the heterogeneous permeability and h^{ϵ} is the heterogeneous pressure (or head); superscript ϵ indicates micro-scale properties in microscopic scale. The Koebbe's upscaling or average processes apply the periodic perturbation method to find equivalent permeability in macro-scale domain, which converts the elliptic flow equation 3.27 into the form

$$- \nabla \cdot v^{\#} = f \tag{3.28}$$

where $v^{\#} = - K^{\#} \nabla h^{\#}$. In the equation, $K^{\#}$ is the homogenized permeability in macroscopic scale and $h^{\#}$ is the pressure variable (or head) in macroscopic domain; superscript # indicates macro-scale properties. The technique was extended from the work by Bourgeat (1984) and Amaziane and Bourgeat (1988), which also applied a perturbation method to solve the equations governing flow through porous media.

For the two-dimensional fractured domain shown Figure 3.5, the fracture map was initially discretized to produce 200×200 grid blocks. Homogenization was performed to produce coarse grids (10×10, 20×20 and 40×40 grids) with equivalent permeabilites. The result of permeability upscaling using the homogenization method for the test fracture map (Figure 3.5) is illustrated in Figure 3.20, where the absolute matrix permeability was assumed to be 1 *mD* while the fracture permeability was 100 *mD.* The homogenization method usually produces a full nine component permeability tensor. Since typical black-oil reservoir simulators are not capable of accepting full tensor representations, off-diagonal terms were suppressed in the current implementation. With the equivalent permeability generated by the homogenization method, water flood simulations were performed using the finite difference simulator developed by the

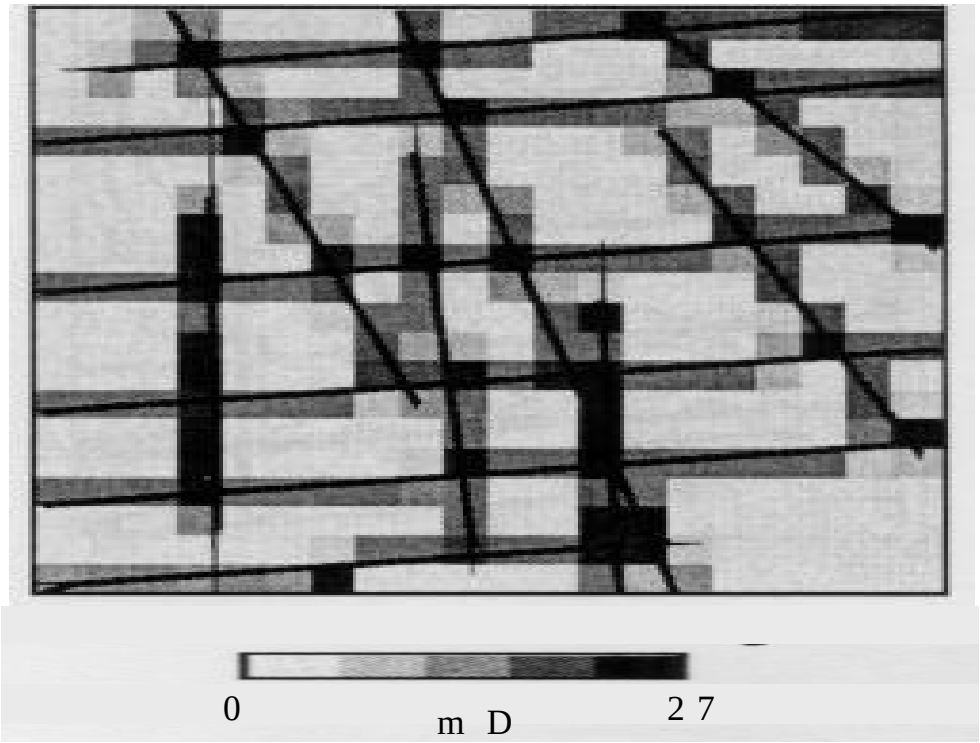


Fig 3.20 The equivalent permeability magnitude of upscaling the fracture network from 200×200 grid blocks to 20×20 grid blocks for a permeability contrast 100. The absolute matrix permeability was 1 mD and the fracture permeability was ten times the matrix permeability.

Computer Modeling Group (CMG). For the 10×10 resolution, water saturation profiles for permeability contrasts of 10 and 100 are shown in Figure 3.21 and 3.22. The flow rate in these simulations was 0.014 m^3/day. As expected, the profiles are more or less symmetric for the low contrast. The water saturation profiles at 0.32 m^3/day for the high permeability contrast (1000 to 1) are shown in Figure 3.23. Recoveries at 1.4 PV total injection are presented in Table 3.4 for various homogenization methods at different resolutions. Recoveries even for the 40×40 resolution are greater than those predicted in the discrete fracture model. Water saturation profiles for the 40×40 resolution at 0.7 PV injection are shown in Figure 3.24. At this fine resolution, the homogenization begins to capture the features of the network better; however, the results are still vastly different from the discrete fracture model.

Arithmetic, geometric and harmonic average methods of homogenization were also performed (Koebbe, 1998) to generate the equivalent permeability and profiles were generated at identical flood conditions. For the high permeability contrast, the water saturation profiles for the arithmetic, geometric and harmonic methods of generating equivalent properties are shown in Figure 3.25, 3.26 and 3.27. All the floods in these simulations were performed at the high injection rate of 0.32 m^3/day. Comparison of the profiles shows that in terms of being able to capture the features of the fracture network, Koebbe's homogenization (1998) is the best.

3.4 Conclusions

A new discrete-fracture model that incorporates fractures spatially explicitly was developed and tested on a two-dimensional data set. A diagonally oriented water flood was

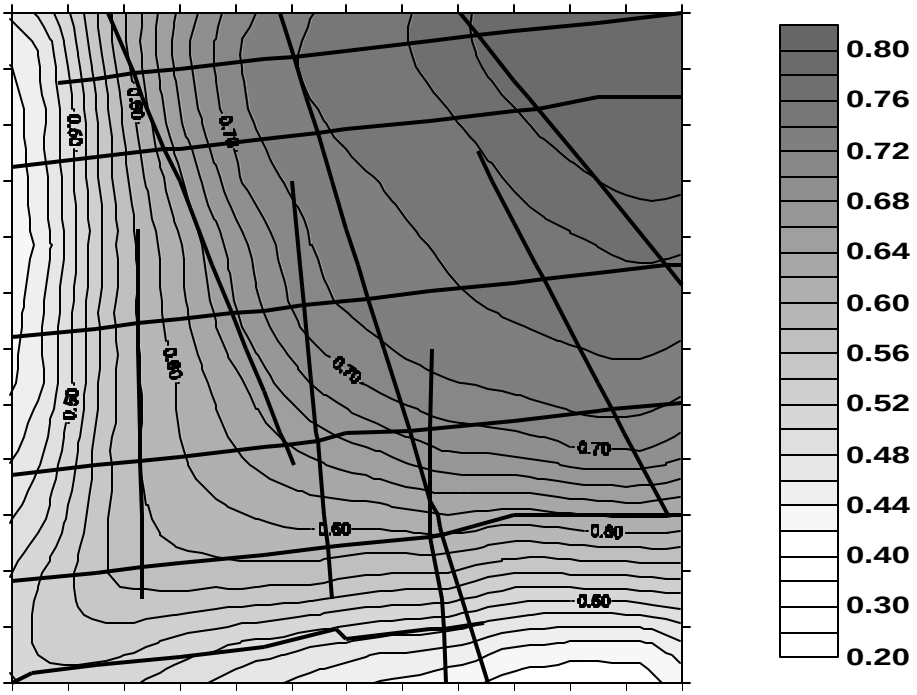


Figure 3.21 Water saturation profiles for the homogenized model: permeability contrast of 10, flow rate of 0.014 m^3 per day, and resolution 10 by 10.

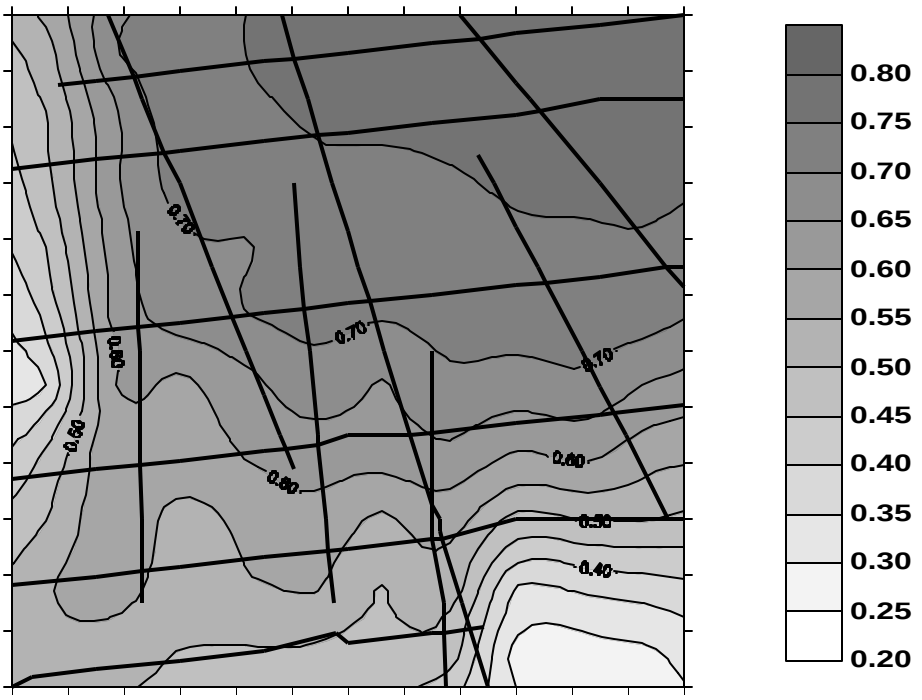


Figure 3.22 Water saturation profiles for the homogenized model : permeability contrast of 1000, flow rate of 0.014 m^3 per day, and resolution 10 by 10.

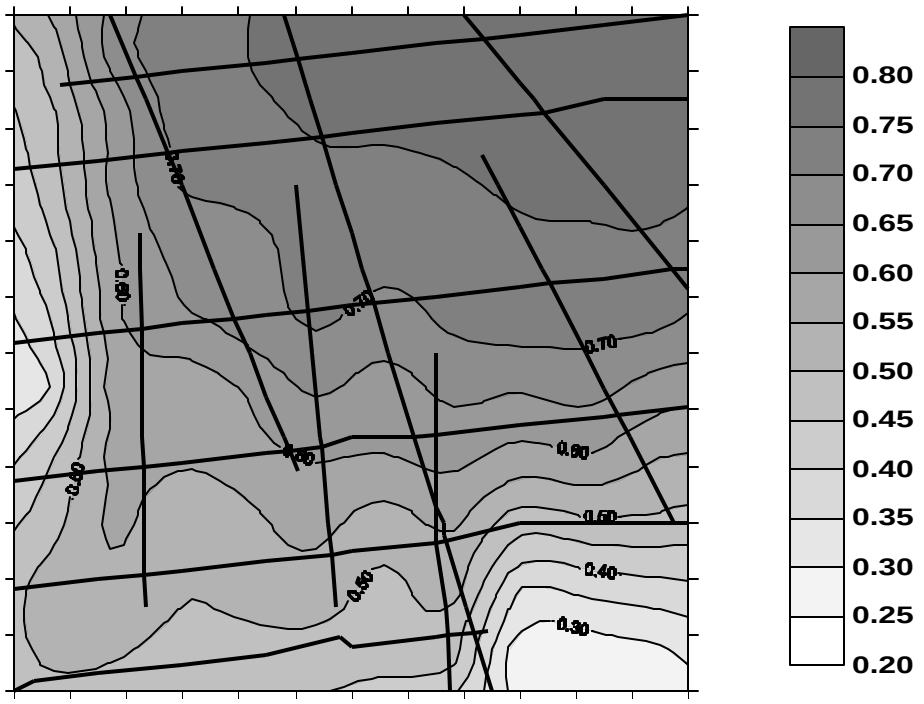


Figure 3.23 Water saturation profiles for the homogenized model: permeability contrast of 1000, flow rate of 0.32 m^3 per day, and resolution 10 by 10.

Table 3.4 Percentage oil in place (OOIP) recovered after injecting 1.4 pore volumes of water (Equivalent Property Model).

Model	Perm. Contrast	Resolution	Rate (m^3/day)	Recovery % OOIP
Homogenization	10 to 1	10×10	0.014	64
Homogenization	100 to 1	10×10	0.014	64
Homogenization	1000 to 1	10×10	0.014	59
Homogenization	1000 to 1	10×10	0.32	63
Homogenization	1000 to 1	20×20	0.32	58
Homogenization	1000 to 1	40×40	0.32	56
Arithmetic upscaling	1000 to 1	10×10	0.32	59
Geometric upscaling	1000 to 1	10×10	0.32	59
Harmonic upscaling	1000 to 1	10×10	0.32	59

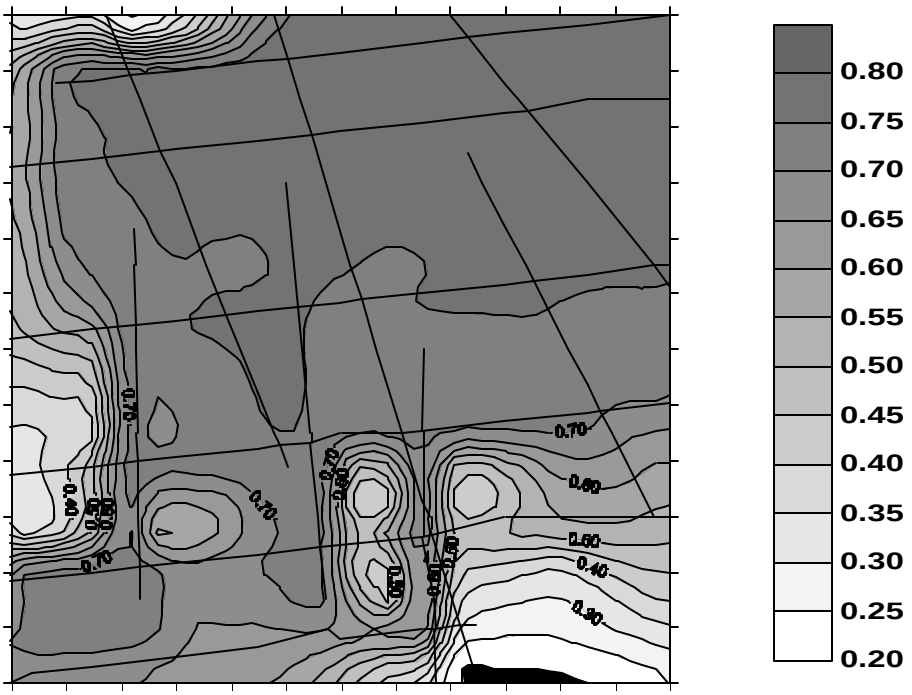


Figure 3.24 Water saturation profiles for the homogenized model : permeability contrast of 1000, flow rate of 0.32 m^3 per day, and resolution 40 by 40.

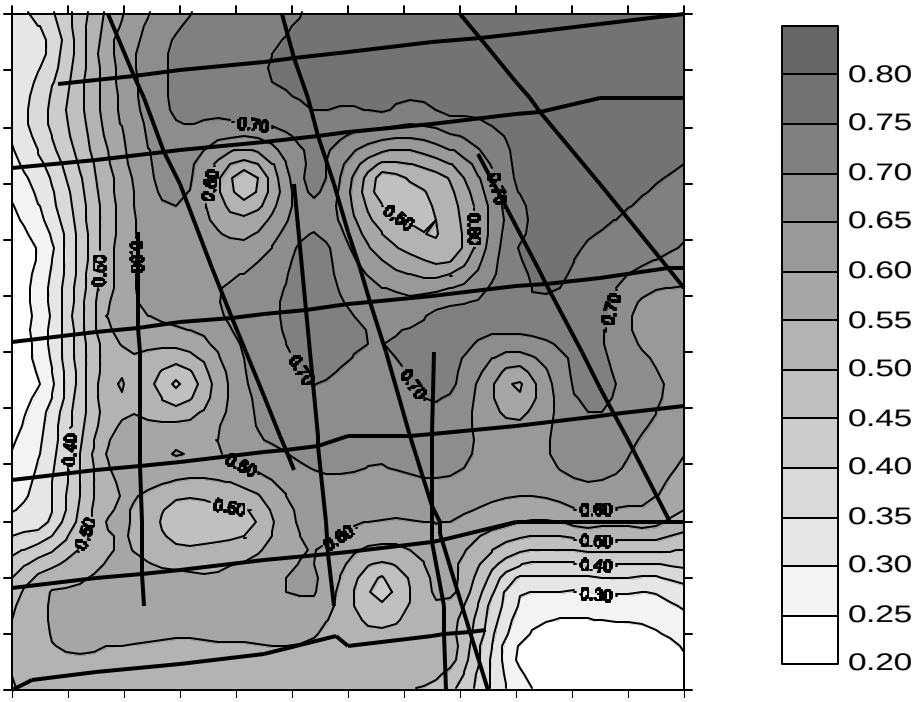


Figure 3.25 Water saturation profiles for the model where the equivalent properties were generated using the arithmetic average: permeability contrast of 1000, flow rate of 0.32 m^3 per day, and resolution 10 by 10.

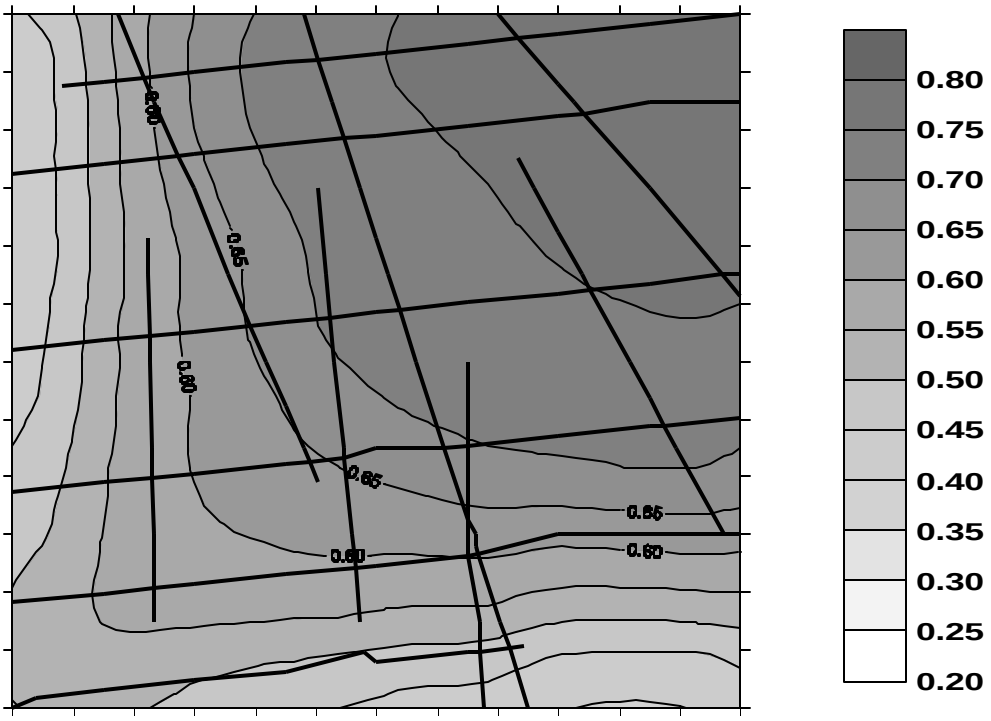


Figure 3.26 Water saturation profiles for the model where the equivalent properties were generated using the geometric average: permeability contrast of 1000, flow rate of 0.32 m^3 per day, and resolution 10 by 10.

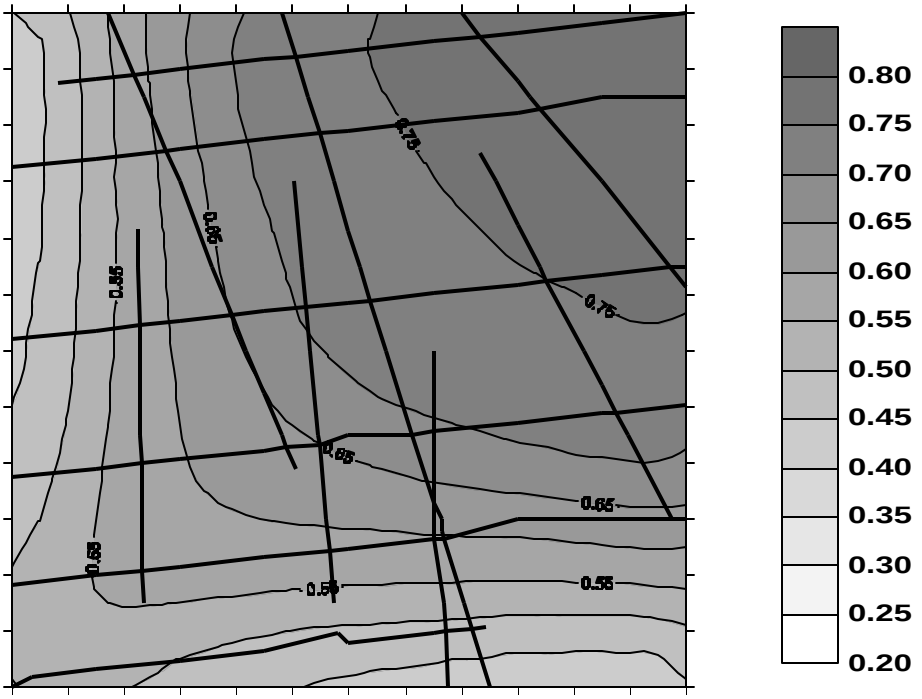


Figure 3.27 Water saturation profiles for the model where the equivalent properties were generated using the harmonic average: permeability contrast of 1000, flow rate of 0.32 m^3 per day, and resolution 10 by 10.

simulated. The nonfractured, finite element model was validated using a finite difference, commercial black-oil simulator. For a single-fracture system, the discrete-fracture model produced results comparable to the model with an explicit finite element discretization of the domain.

The discrete-fracture model was flexible, in that it was possible to use different sets of properties to represent the matrix and the fracture systems, and different portions of the domain. Several sets of simulations were performed to assess the impact of a number of input parameters. As expected, matrix to fracture permeability contrast played a dominant role in determining oil recovery. At high contrast (1:1000), there was considerable oil bypassing and reduced oil recovery. At higher injection rates, the bypassing was even more pronounced. As the matrix absolute permeability increased, the fracture network played a less dominant role. Reducing the capillary pressure in the fractures generally increased oil recovery and had more significant effect when the matrix permeability was higher. There was the significant difference of water saturation profiles between homogenization methods and discrete fracture model. Any homogenization method could not capture the oil bypassing observed in the discrete fracture model. As the permeability contrast is reduced, the match between the homogenization and the discrete fracture model improves. But higher oil recovery values were obtained in the simulations of all homogenization models.

The discrete-fracture model developed in this paper is an alternative to conventional dual-porosity (dual-permeability) or single-porosity models and can be used when detailed fracture network information is available. It allows a much finer examination of the multiphase displacement processes in fractured porous media than has been possible

through conventional models where fractures are represented by uniform orthogonal networks or averaged out into single statistical property (permeability).

CHAPTER 4

INEXACT NEWTON-KRYLOV METHODS FOR THE SOLUTION OF IMPLICIT RESERVOIR SIMULATION PROBLEMS

4.1 Introduction

Implicit Pressure Explicit Saturation (IMPES) solution scheme has been a popular numerical procedure for black oil simulations in a computational point of view and is applicable under most simulation conditions. However, IMPES does have stability limitations, particularly with regard to certain capillary pressure functions and grid block geometries (aspect ratio). As a consequence, for certain types of capillary pressure-saturation functionalities and aspect ratios, extremely small time steps would be required for a stable solution in IMPES (Aziz and Settari, 1979). Thus an implicit formulation is necessary to address the problem of stability in solving the set of partial differential equations. When a fully implicit scheme converts the partial differential equations of black oil reservoir simulation to algebraic equations, usually a set of nonlinear algebraic equations result.

$$F(x) = 0 \tag{4.1}$$

The number of equations is N (grid blocks) $\times$ the number of phases and the set is to be solved at every time step of the simulation. The method of choice for the solution of these nonlinear equations is the Newton's method that is used in most conventional implicit formulations. Briefly, for a given initial guess x_0, the Newton's method computes a sequence of solution increments s_k and iterates x_k as follows:

$$\begin{aligned} &\textit{Until} \quad \textit{convergence} \\ &\textit{Solve} \quad F'(x_k)s_k = -F(x_k) \\ &\textit{Set} \quad x_{k+1} = x_k + s_k \end{aligned} \tag{4.2}$$

Although Newton's method is attractive because it converges rapidly from any sufficiently good initial guess, one drawback of the method is to solve the equations at every Newton's iteration. Computing the exact solution of the linear matrix system of equation 4.2 at each stage of the Newton iteration can be very expensive if the number of unknowns is large and the initial guess is far from the solution. Therefore, application of the inexact Newton's method was examined. The nonlinear equations for this application were generated by a fully implicit finite element discrete fracture model. The differential equations, the finite element formulation and the set of nonlinear equations are discussed in detail in Chapter 3. Note that the main idea of inexact Newton's scheme is the inexactness of the solution increments obtained by solving the linear system in the Newton's formulation (4.2). In the implementation of the inexact Newton method, the transpose free quasi-minimal residual method (TFMQR) was used to find approximations

to the Newton's iterates. TFQMR, when implemented with a certain inexact tolerance, reduces the computational cost of the overall inexact scheme compared to the exact Newton's method. TFQMR was used since it turned out to be the most efficient of the linear solver as shown in Chapter 2.

4.2 The application of inexact Newton method to a fully implicit oil reservoir simulation

To understand the inexact Newton's algorithm, it is important to define the so-called enforcing terms. The linear system from Newton's equation 4.2 is solved only approximately in the inexact formulation. The general inexact formulation is shown below:

$$\left\| F(x_k) + F'(x_k)s_k \right\| \le h_k \left\| F(x_k) \right\| \tag{4.3}$$

This equation essentially indicates an inexact step to obtain only an approximate solution of the linear equation to specified accuracy h_k. Thus, the inexact step saves the most costly part of computing the exact solution in Newton's method and also offers great opportunity for exploiting parallelism when used with efficient iterative linear solution methods. In the study of the inexact Newton's framework to oil reservoir simulation, the following particular enforcing terms were examined (Eisenstat and Walker, 1996).

1. $h_k = 10^{-4}$, the choice which requires uniformly close approximations of Newton steps for all iterations and results in fast local convergence rate per Newton iteration, which can considered close to a exact Newton's formulation.

2. $h_k = \left(\frac{1}{2}\right)^{k+1}$, the choice solving relatively inaccurate approximations of Newton steps with small k and with no information about $\|F(x_k)\|$.

3. $h_k = \min\{\frac{1}{(k+2)}, \|\|F(x_k)\|\}$, the choice also allowing relatively inaccurate approximations of Newton method with small k and with some information about $\|F(x_k)\|$. Its also depends on the scale of $\|F(x_k)\|$.

4. $h_k = \frac{\left|\left\|F(x_k)\right\| - \left\|F(x_{k-1}) + F'(x_{k-1})s_{k-1}\right\|\right|}{\left\|F(x_{k-1})\right\|}$, the choice reflecting the agreement between $\|F(x_k)\|$ and its local linear model at the previous step. In this choice, the initial enforcing term was set at 0.5, which can help to avoid oversolving the Newton equations when a significant disagreement between $\|F(x_k)\|$ and its local linear model results.

For practical effectiveness, a backtracking algorithm (Press et al., 1992) is used, which offers global convergence. If $\|F(x_k+s_k)\|$ is not acceptable, the algorithm will backtrack to try an acceptable $\|F(x_k+qs_k)\|$, as shown in Figure 4.1.

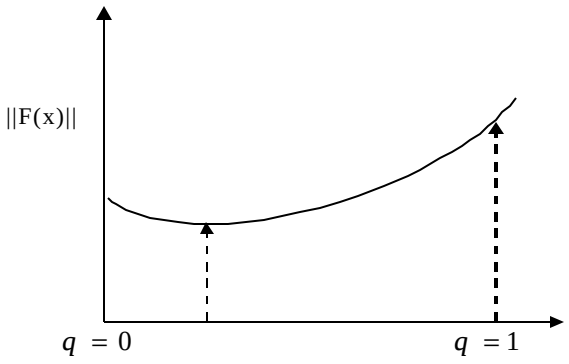


Fig 4.1 Quadratic backtracking scheme used in the inexact Newton algorithm

In the strategy for backtracking, a new functional value is defined, $f(q)=\|F(x_k+qs_k)\|$, the one-dimensional restriction of $\|F(x_k)\|$ to the line through x_k in the direction s_k. If it needs to backtrack, a quadratic line search model will be used to minimize the current information of $f(q)$. Initially, two pieces of information are known about $f(q)$,

$$f(0) = \|F(x_k)\| \tag{4.4}$$

$$f'(0) = F'(x_k)F(x_k) \cdot s_k \tag{4.5}$$

After calculating $\|F(x_k+s_k)\|$, another functional value is known,

$$f(1) = \|F(x_k + s_k)\| \tag{4.6}$$

If $f(1) \geq f(0)$, it models $f(q)$ by a one dimensional quadratic model satisfying the above conditions,

$$f(q) \cong [f(1) - f(0) - f'(0)]q^2 + f'(0)q + f(0) \tag{4.7}$$

q is calculated by:

$$q = -\frac{f'(0)}{2[f(1) - f(0) - f'(0)]} \tag{4.8}$$

The q thus calculated is the new value employed in the inexact method. In the implementation of the algorithm, $t = 10^{-4}$ was used to ensure sufficient reduction in the backtracking step. The final algorithm used here is shown in Figure 4.2, which was developed by Eisenstat and Walker (1996). Based on the algorithm, the inexact Newton algorithm was applied to a two-phase (oil and water phases) oil reservoir simulation in a finite element formulation using linear triangular elements. The two flow equations (for oil and water phases) described in the previous chapter were expanded in the two primary variables (oil pressure and water saturation). The iterations in the implementation are:

1. Estimate the primary variables, Po (oil pressure) and Sw (water saturation), which might be based on the values at the previous time step
2. Calculate the residual vector and the Jacobian matrix based on current approximations. As was shown in the previous chapter, the residual vector and Jacobian matrix were calculated in matrix form

To solve $F(x) = 0$.

1. Set x_o to solve $\|F(x_k)\|$ for $k = 1,2\ldots$
2. Chose initial η_k and s_k such that $\left\|F(x_k) + F'(x_k)s_k\right\| \le h_k \left\|F(x_k)\right\|$

While $\|F(x_k+s_k)\| > (1- t + t\,\eta_k)\|F(x_k)\|$

Chose θ in (0.1,0.5)

Update $s_k \leftarrow \theta s_k$ and $\eta_k \leftarrow 1 - \theta\,(1 - \eta_k)$

3. Set $x_{k+1}=x_k+s_k$
4. Check tolerance

Figure 4.2 The inexact Newton method with backtracking scheme.

$$residual(1) = AoP_o + Bo1\frac{\partial S_w}{\partial t} + Bo2\frac{\partial P_o}{\partial t} - Q_o - Go \tag{4.9}$$

$$residual(2) = Aw_{pc}S_w + AwP_o + Bw1\frac{\partial S_w}{\partial t} + Bw2\frac{\partial P_o}{\partial t} - Q_w - Gw \tag{4.10}$$

$$Jacobian = \begin{bmatrix} \frac{\partial residual(1)}{\partial S_w} & \frac{\partial residual(1)}{\partial P_o} \\ \frac{\partial residual(2)}{\partial S_w} & \frac{\partial residual(2)}{\partial P_o} \end{bmatrix} \tag{4.11}$$

All the coefficients were identified in the previous chapter.

3. Solve the linear systems of equation to make corrections to the current approximations by using the appropriate enforcing term in the inexact Newton algorithm.

$$\begin{bmatrix} \frac{\partial residual(1)}{\partial S_w} & \frac{\partial residual(1)}{\partial P_o} \\ \frac{\partial residual(2)}{\partial S_w} & \frac{\partial residual(2)}{\partial P_o} \end{bmatrix} \begin{bmatrix} \partial S_w \\ \partial P_o \end{bmatrix} \approx \begin{bmatrix} -residual(1) \\ -residual(2) \end{bmatrix} \tag{4.12}$$

4. Updating the primary variables with the new information from Newton's iteration.

$$S_w = S_w + \partial S_w,\ P_o = P_o + \partial P_o \tag{4.13}$$

4.4 Numerical results

A two-phase flow problem of two-dimensional finite element formulation described in the previous chapter was solved by the four different inexact Newton's schemes. A 18.3 m by 18.3 m two-dimensional rectangular reservoir model was used for the numerical experiment. There were a total 441 nodes and 800 triangular elements. The computational times for the five time steps were compared for all the methods. The effect of the choice of enforcing terms on computational time is shown in Table 4.1. As can be seen from the table, even though choice 4 takes more Newton iterations, computational time is significantly less. The convergence behavior for the first and the fourth choices is shown in Figure 4.3, in which the logarithm of $\|F(x)\|$ is plotted as Newton iterations progress. The two methods (choice 1 and 4) show smooth convergence behavior without convergence failure. Enforcing term choice 4 takes more iterations because it used larger η in most iterations. The parallel performance of the program was examined in the next chapter on two parallel machines (SGI Power Challenge and SGI Origin 2000) using choice 4 for the enforcing term.

Table 4.1 Computational times for the four choices of enforcing terms examined in this study

Enforcing terms	cpu time (sec)	Newton iterations	Linear solver iterations	Tolerance
1	1174	15	228	1.085E-05
2	955	15	180	1.098E-05
3	1744	15	367	1.109E-05
4	521	19	77	7.984E-06

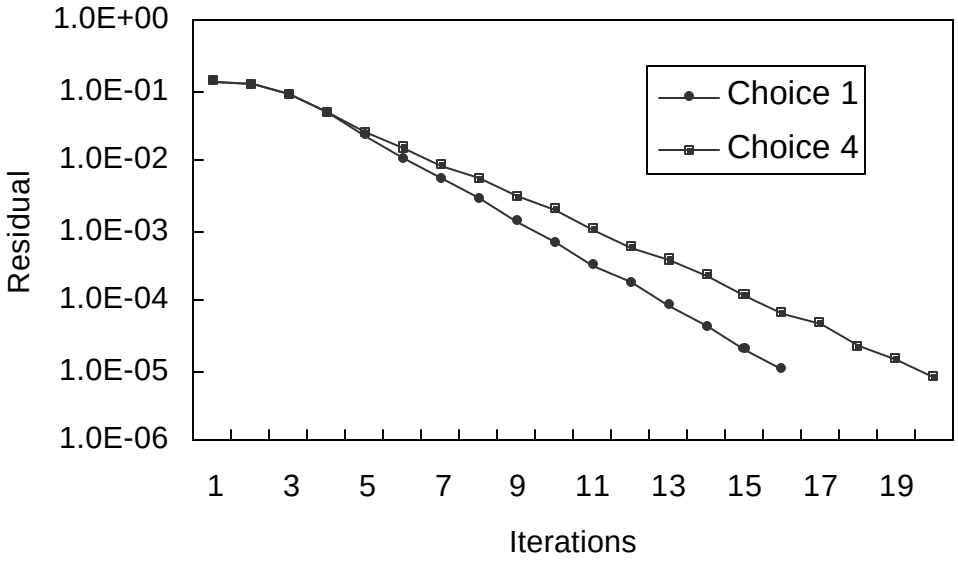


Fig 4.3 Convergence behavior for two of the inexact Newton formulations

Significant acceleration is achieved through domain decomposition method. The parallel implementation of the scheme is provided in the next chapter.

4.4 Conclusions

All the inexact Newton choices studied were practically convergent with highly desirable superlinear rates of convergence. The essential difference in the four methods studied was the enforcing term that decided the level of inexactness of the method. The method designed to reflect the agreement between the function and its local linear model at the previous time step was found to be significantly better than methods that arbitrarily fixed the level of inexactness (about 50 % saving in CPU time compared with uniformly close approximations of Newton method). The conclusions of this study would be applicable to any set of nonlinear equations resulting from implicit reservoir simulation.

CHAPTER 5

PARALLEL IMPLEMENTATION OF THE FINITE-ELEMENT, DISCRETE FRACTURE MODEL

5.1 Introduction

Standard commercial oil reservoir simulators are based on finite difference approximations in time and space and can be designed to achieve high performance on today's parallel computers (Killough, 1993). The explicit fracture representation in simulating fractured reservoirs would require an exceptionally fine grid and would make practical implementation with even a few fractures nearly impossible. In Chapter 3, a discrete fracture model was developed as an alternative to conventional dual continua models and to fine-grid, explicit fracture model. The finite element solution to the problem is also computationally challenging and parallel implementation would be mandatory for the solution of realistic problem sets. The primary objective of this chapter is to examine the performance of the discrete fracture model on parallel machines using the domain decomposition technique. Domain decomposition is the most commonly used technique for parallelizing reservoir simulators. In oil reservoir simulation, most of the reported work has been with respect to finite difference simulators. Even though the development of parallel black oil (Hemaanth-Kumar and

Young, 1991) and compositional simulators (Killough, 1995, Killough and Rao, 1991, Rutledge et al., 1991 and Rame and Delshad, 1995) have been reported in the literature, the parallel computing application to finite element models has not been reported especially for fractured oil reservoir system. In finite element formulations, the grid is usually unstructured and as a result the computational kernel consisting the parallel implementation scheme for the solution of linear and nonlinear equations is different from the one in finite difference discretization. In order to solve the linear and nonlinear equations, parallel iterative solvers discussed in Chapter 2 and 4 are employed. More details of the finite element equations for two-phase and two-dimensional discrete fracture model were presented in Chapter 3. The following sections describe the parallel implementation strategy using domain decomposition method. Parallel performance analyses were undertaken for four problem stets on SGI Power Challenge (shared memory machine) and SGI Origin 2000 (distributed shared memory machine). Results of these analyses are presented at the end of the chapter.

5.2 Parallel implementation

When the multiphase, discrete fracture flow equations 3.1 ~ 3.4 are solved implicitly, the solution was ultimately obtained by the sets of nonlinear ordinary differential equations 4.9 and 4.10. Every step in the solution of the equations involves the solution of a linear matrix system 4.11, which is a Newton-like iteration scheme. In this method, the correction vector of primary variables P_o (oil pressure) and S_w (water saturation) minimizes the norm of residual equations 4.9 and 4.10. In the parallel

implementation of the solution procedures mentioned above, the most expensive part is the solution of the large sparse linear system of equation 4.11.

The linear matrix system of equation 4.11 was solved in parallel by first decomposing the problem domain geometrically. Here, the finite elements and nodes are distributed over multiple processors. Obviously, the element-by-element computation of equations 3.7, 3.8 and 3.9 (an assembled global matrix system) is a generic parallel-processing problem. That is because finite element method consists of breaking up the computational domain into the elements, in which the governing equations are discretized using low-order polynomial shape functions. Indeed, elemental contribution to a assembled global matrix system can be calculated independently on individual processors (Figure 5.1).

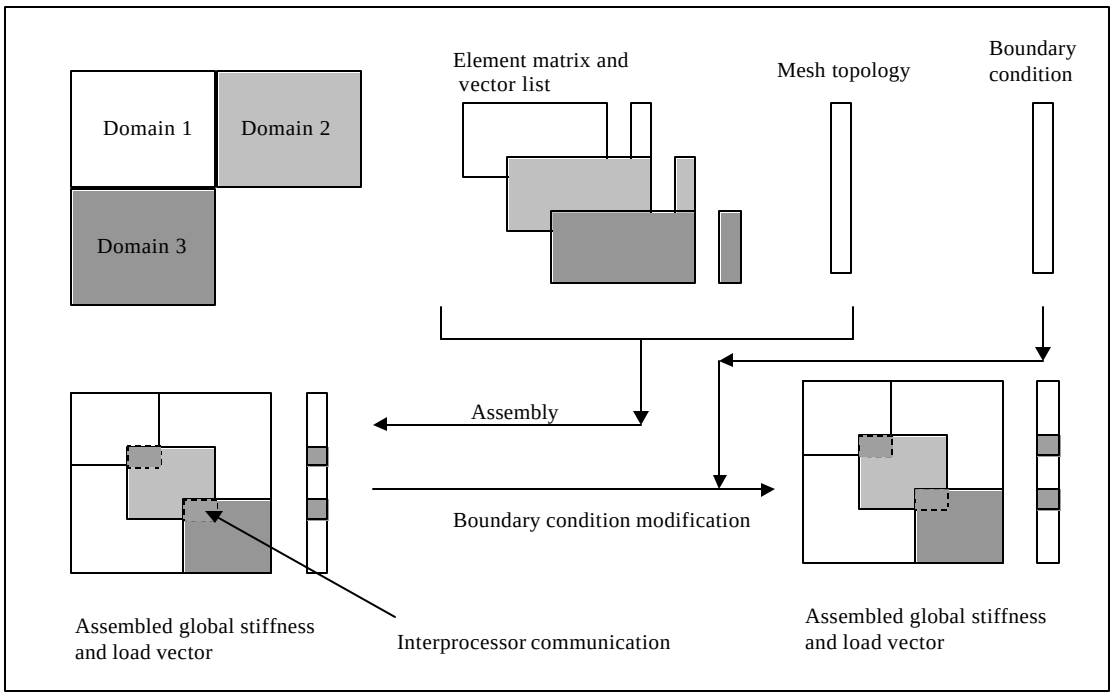


Figure 5.1 Domain decomposition strategy in finite element computations.

In the figure, the globally assembled linear matrix system requires appropriate interprocessor communications for the boundary regions among processors. To reduce the load level of interprocessor communications, shared finite elements are allocated to the boundary of each processor. In this scheme, each processor works only on its subdomain to construct a global linear matrix system, which is distributed on multiple processors. It requires no communications in the stage of construction for a global linear matrix system. Once the linear system is solved, the new information of updated primary variables is exchanged between processors. This concept is illustrated for an 18 node, 20 triangular element system in Figure 5.2. In this figure the domain is divided among two processors.

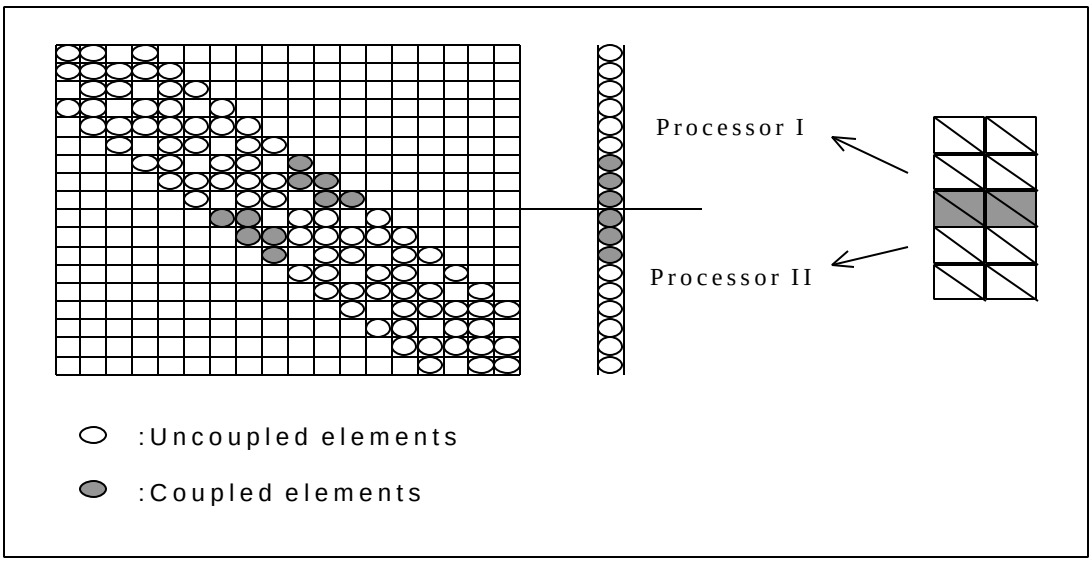


Figure 5.2 The structure of linear matrix system of equations in matrix form divided among two processors; the system comprises of 18 nodes and 20 triangular elements, as shown.

Each processor shares four elements (three shared nodes). The figure shows that the coupled matrix elements were constructed by the contributions of triangular elements shared between the processors, which requires the communication of the three-coupled vector elements. The communication for the three-coupled vector elements is required only after the linear matrix system is solved in the iterations of Newton-like method (equation 4.2 or 4.11). In constructing linear matrix system, each processor has already the information about the shared elements. Thus, no interprocessor communication is required between the processors in the construction stage of the linear matrix system. From the domain decomposition, each processor solves the nine-by-nine linear matrix system from 14 triangular elements and 9 nodes; each vector element has two primary variables (oil pressure, P_o and water saturation, S_w), which makes one matrix element contain four-by-four sub-system. The domain decomposition for the parallel implementation used here is similar, in principle to the scheme described by Keunings (1995).

The Fortran program for the implementation of this idea is written in the following sequence.

- At the beginning of calculations, each processor is allocated elements of a subdomain where a dummy index is given for the shared elements at the interface boundary of each processor.
- All processors then perform in parallel the sequential finite element matrix construction for its own domain.
- When linear and nonlinear solution equations are solved, matrix and vector calculations are achieved through the parallel communication at the interface

boundary of shared elements among the processors, which makes each processor hold updated interface variables.

- Once the updated interface variables are available, the processors can compute the next iteration concurrently for its own subdomain.

The parallel communication for the interface boundary among the processors consists of two steps: moving the data of the indexed shared elements into temporary arrays (buffers) and sending and receiving the data among the processors using the message passing library. The manner in which the information of interface elements is communicated is a key issue as far as parallel efficiency is concerned. In the study the non-blocking message passing operations of MPI library (Gropp et al, 1995) are used as follows.

```
 SUBROUTINE SEND_MINUSB
&          (M, M_LOCAL , M_GLOBAL , M_COMM  , N_SEND  ,
&              ME      , ME_MINUS , TEMP)

 INCLUDE 'mpif.h'

 INTEGER  STATUS(MPI_STATUS_SIZE,1), info, req(1)

 DOUBLE PRECISION TEMP(M_LOCAL)

 INTEGER I, J, M, ME, ME_MINUS

 INTEGER M_LOCAL, M_GLOBAL(M), M_COMM(M,4)

 INTEGER N_SEND

 DOUBLE PRECISION  TEMP_SEND_MINUS(N_SEND)

 J=0
 DO I=1,M_LOCAL
    IF (M_COMM(I,1) .NE. 0) THEN
        J= J +1
        TEMP_SEND_MINUS(J)= TEMP(I)
    ENDIF
 ENDDO

 call MPI_ISEND(TEMP_SEND_MINUS, N_SEND, MPI_DOUBLE_PRECISION,
&              ME_MINUS, 0    , MPI_COMM_WORLD,
&              req(1), info)

 call MPI_WAITALL(1,req,status,info)
```

```
 RETURN
 END

 SUBROUTINE RECEIVE_PLUSB
&          (M, M_LOCAL , M_GLOBAL , M_COMM , N_RECEIVE ,
&              ME      , ME_PLUS  , TEMP)

 INCLUDE 'mpif.h'

 INTEGER  STATUS(MPI_STATUS_SIZE,1), info, req(1)

 DOUBLE PRECISION TEMP(M_LOCAL)

 INTEGER I, J, M, ME, ME_PLUS

 INTEGER M_LOCAL, M_GLOBAL(M), M_COMM(M,4)

 INTEGER N_RECEIVE

 DOUBLE PRECISION  TEMP_RECEIVE_PLUS(N_RECEIVE)

 call MPI_IRECV(TEMP_RECEIVE_PLUS   , N_RECEIVE,
&              MPI_DOUBLE_PRECISION , ME_PLUS  ,
&              0                    , MPI_COMM_WORLD,
&              req(1)  , info)

 call MPI_WAITALL(1,req,status,info)

 J=0
 DO I=1,M_LOCAL
    IF (M_COMM(I,4) .NE. 0) THEN
        J= J +1
        TEMP(I)= TEMP_RECEIVE_PLUS(J)
    ENDIF
 ENDDO

 RETURN
 END
```

Here, `M_LOCAL`, `M_GLOBAL` and `M_COMM` are the dummy indices to distinguish the information of the coupled vector elements that require interprocessor communications between `ME` and `ME_PLUS(or ME_MINUS)`. The array, `TEMP(M_LOCAL)`, stores the information to be exchanged between processors. The basic features of MPI library are provided in Appendix.

5.3 Numerical results

5.3.1 Model problems

Numerical simulations were performed on four problem sets. Physical dimensions of the problem sets were 60 feet by 60 feet. In the first set, which considered a fractured system, there were a total of 193 nodes, 341 triangular elements with 139 fracture elements. The fracture map was created using the FracMan^{TM} software of Golder Associates, Inc. Details of the geologic data input and the methodology of fracture network generation are discussed in detail in Forster, et al. (1998). Detailed initial fracture map generated using field data is shown in Chapter 3 (Figure 3.4). Other test problems were three non-fractured systems:

- Problem set 2): 441 nodes and 800 triangular elements
- Problem set 3): 676 nodes and 1250 triangular elements
- And problem set 4): 784 nodes and 1450 triangular elements

There was an injection well at the northeast corner and a production well at the southwest corner, diagonally opposing the injection well. Injection and production rates were identical at 0.014 m^3/day and computational times for a total of 10 days of simulations were compared. Fracture to matrix permeability contrast used for the problem set 1) was 100 and the absolute matrix permeabilities for all the test problems were 200 mD. Other parameters are given in Table 3.1. Different from finite difference grid system, parallel finite element scheme causes the unbalanced communication overhead from the irregular domain distributed on multiple processors. By hand calculation, the study attempted to form clusters of finite elements that minimize the surface boundaries contacted between subdomains. The domain decompositions are shown in Figures 5.3 and 5.4 for the

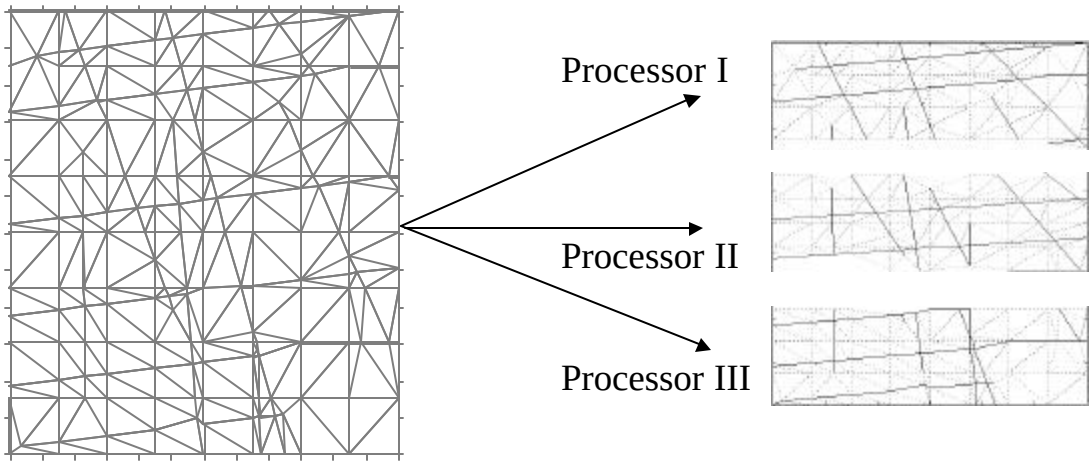


Figure 5.3 The discrete fracture domain decomposed among three processors (193 nodes system of 341 triangular elements and 139 fracture lines).

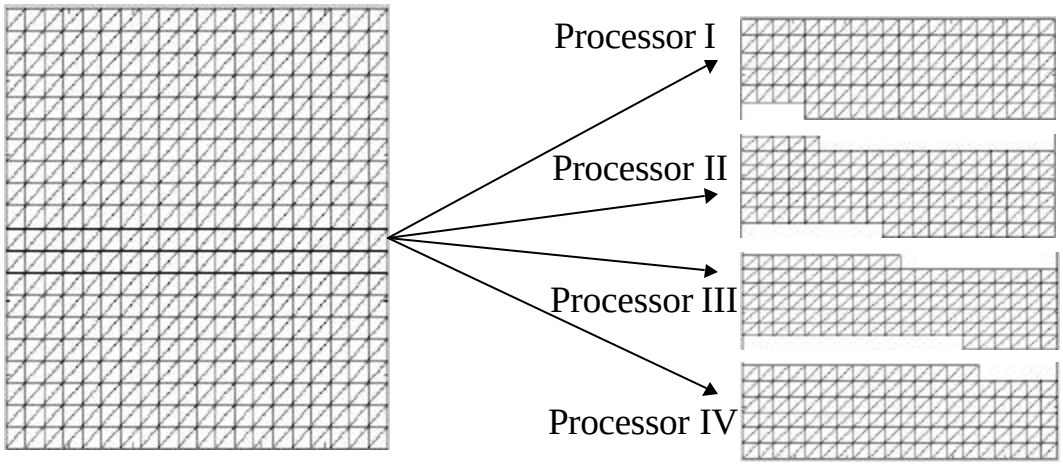


Figure 5.4 A 441 nodes, 800 elements system decomposed among four processors.

problem set 1) and 2). Table 5.1 shows the number of nodes and elements distributed on four processors for the problem set 4.

5.3.2 Performance of the parallel program

The performance of the parallel program was evaluated on SGI power challenge (a four-processor shared memory machine). The processors employ the MIPS R8000 chip, with 64-bit registers and a cycle time of 75 M Hz. Table 5.2 shows parallel speedup and computational time of each subroutine obtained for the problem set 1). The linear solver consumed significant CPU time ranging from 33 % to 60 % of total CPU time, which indicate that the speedup observed resulted from the use of efficient linear and nonlinear solvers based on domain decomposition scheme. The overall computational time for the solution of linear matrix system remains about the same, which is resulted from the more iterations required for local linear solver; some information is lost by the coupled matrix elements. The change in computational time is significant for setting up the coefficient matrices and vectors. Overall speed up factor of 1.8 with 4 processors was obtained. Here, speedup is defined as the ratio between the total computational time for single processor and that observed with P number of processors.

Performance of the parallel program was also studied on SGI Origin 2000. The Origin 2000 is a scalable distributed shared memory architecture that provides a follow on to the SGI Power Challenge class of symmetric multiprocessing systems. Test problems identified previously were executed on the Origin. For the model problem with 193 nodes, the results were comparable to those for SGI Power challenge and are

Table 5.1 Distribution of nodes and element for the problem set consisting of a total of 784 node and 1450 triangular elements. This domain decomposition scheme was employed with 8 processors on SGI Origin 2000.

	The number of nodes	The number of elements	The number of shared nodes	The number of shared elements
Processor I	127	190	29	56
Processor II	155	308	57	110
Processor III	155	298	57	110
Processor IV	155	308	57	110
Processor V	155	298	57	110
Processor VI	155	308	57	110
Processor VII	155	310	57	110
Processor VIII	147	234	29	56

presented in Table 5.2. The CPU time decreases significantly, as the number of processors is increased (overall speedup factor of 1.5 with 4 processors).

The computational times required for the other test problems are shown in Table 5.3, 5.4 and 5.5. For these three bigger models, the computational times for the linear solver decreases considerably, as the number of processors is increased. The linear solver continues to be computationally the most expensive component of the simulator. It is also observed that as the number of processors increased, solution of the linear equations became more significant part of the overall computation (Figure 5.5). This is because parallel processing of decomposed domain affects the structure of linear matrix

Table 5.2 Parallel performance of the discrete fracture model on SGI Power Challenge: The test problem consisted of 193 nodes and 341 triangular elements and 139 fracture lines.

	1 Processor	2 Processors	3 Processors	4 Processors
Linear solution (sec)	21	19	21	23
Coefficient (sec)	25	13	9	7
Communication (sec)	0	6.3	10.0	12.0
Overall CPU time (sec)	48	35	33	31

system the most. Especially local solvers lost some information from the domain decomposition (coupled matrix and vector elements). In Figure 5.6, it is seen that communication (message passing overhead) takes up 3~40 % of the total CPU time. Most of communication time is involved in linear and nonlinear solution iterations. The overall speedup behavior is plotted in Figure 5.7. It is observed that as the problem size increased, superlinear performance was obtained. This has occurred due to the memory acess problem (Aziz and Settari, 1979 and Dowd, 1993). Speedup factors of 9.8 (for the problem set 4 with 8 processors), 5.6 (for the problem set 3 with 7 processors) and 4.5 (for the problem set 2 with 4 processors) were obtained as shown in Figure 5.5.

Table 5.3 Parallel performance of a finite element, black oil model on SGI Origin 2000: The problem set consisted of 441 nodes and 800 triangular elements.

	Linear solution (sec)	Coefficient (sec)	Communication (sec)	Overall CPU time (sec)
1 Processor	38	38	0	81
2 Processors	14	14	1.7	31
3 Processors	13	8	4.9	23
4 Processors	11	6	3.8	18
5 Processors	12	5	6.1	18

Table 5.4 Parallel performance of a finite element, black oil model on SGI Origin 2000: The problem set consisted of 676 nodes and 1250 triangular elements.

	Linear solution (sec)	Coefficient (sec)	Communication (sec)	Overall CPU time (sec)
1 Processor	99	81	0	190
2 Processors	53	49	7.0	110
3 Processors	32	21	11.1	58
4 Processors	30	18	15.6	48
5 Processors	22	10.8	10.4	36
6 Processors	22	9	12.6	34
7 Processors	23	8	14.1	34

Table 5.5 Parallel performance of a finite element, black oil model on SGI Origin 2000: The problem set consisted of 784 nodes and 1450 triangular elements.

	Linear solution (sec)	Coefficient (sec)	Communication (sec)	Overall CPU time (sec)
1 Processor	199	153	0	372
2 Processors	67	67	3.2	143
3 Processors	52	37	22.4	97
4 Processors	42	25	22.7	72
5 Processors	39	14	26.2	60
6 Processors	30	13	14.6	46
7 Processors	29	11	15.4	43
8 Processors	26	9	14.5	38

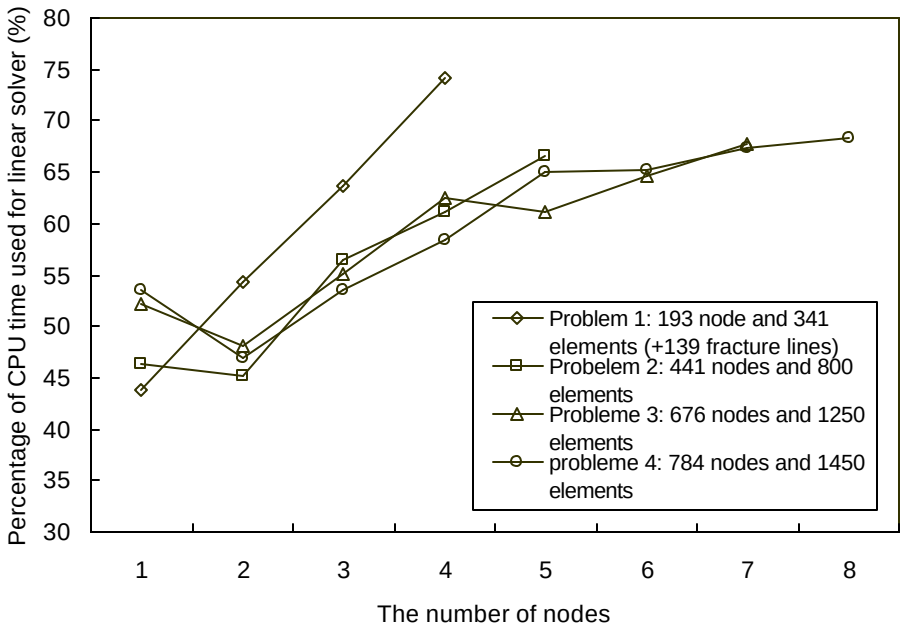


Figure 5.5 Percentage of the total computational time taken up by the linear solver.

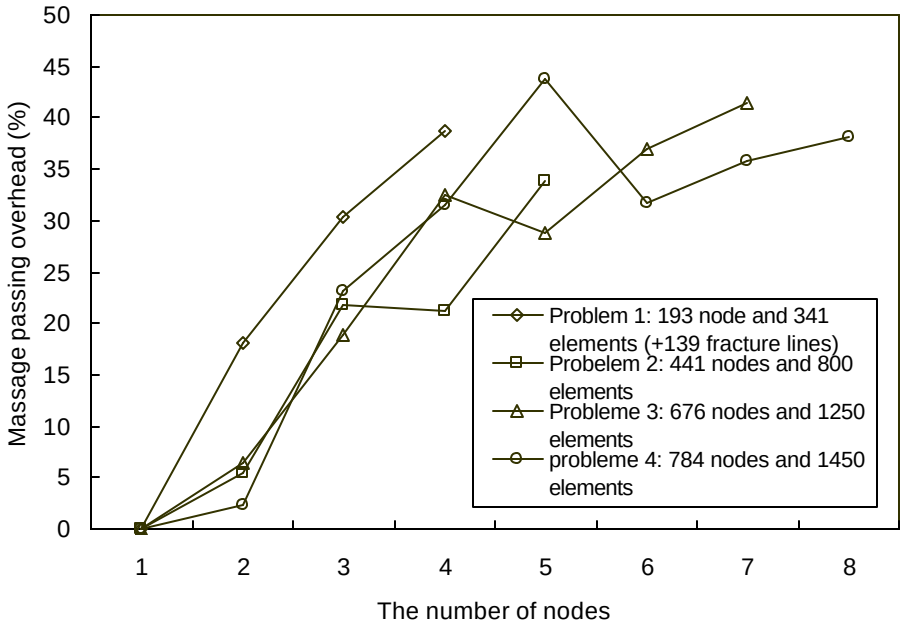


Figure 5.6 Percentage of the total computation time taken up for communication among processors.

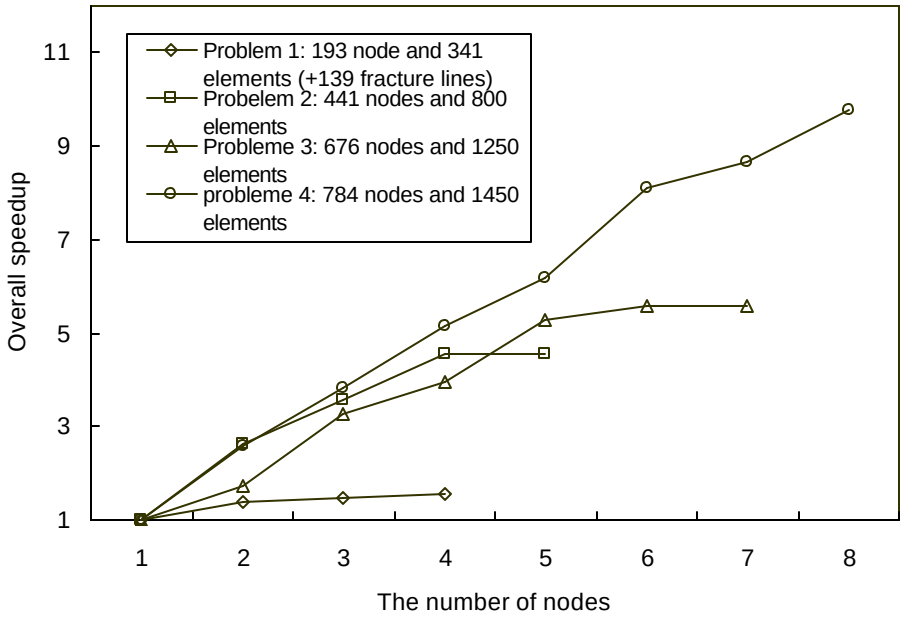


Figure 5.7 Overall speedup obtained for the four different test problems on SGI Origin.

5.4 Conclusions

The new finite element discrete fracture model developed in the previous chapter was parallelized using a simple domain decomposition scheme. In the specific scheme employed, boundary elements between processors were shared. This allowed computation (assembly) of the global matrix concurrently on all processors with parallel communication required only on nodal variables of shared elements. With this parallelization strategy, highly efficient parallel performance was realized on two parallel machines with complete portability. In fact, superlinear performance as observed on both machines with bigger size problems. The superlinear performance is commonly observed where the entire program variables do not fit inside the random access memory of the processor. Despite the simplicity of the decomposition method, the overall parallel performance results are satisfactory.

CHAPTER 6

SUMMARY

The primary objective of this dissertation was to develop computationally efficient discrete fracture model for fractured oil reservoir simulation. To make reservoir simulators computationally efficient two basic strategies can be contemplated.

- Use of efficient linear and nonlinear solvers
- Use of parallel (multiprocessor) computational environment

In this work, the development of efficient linear solvers was undertaken on finite difference discretization of a three-phase black oil model. It was shown that the simpler line-successive over relaxation method (LSOR) is almost as effective in parallel implementation as the sophisticated conjugate gradient method, the transpose free quasi minimal residual method (TFQMR). Parallel implementation of the black oil simulator resulted in reasonable speed up factors on shared and on distributed memory machines.

The idea of using the inexact Newton's method for solving nonlinear equations was also quite successful. When the forcing term incorporated functional information, computational time for one time step computation was essentially cut by half compared to a Newton-like method.

The new discrete fracture model developed in this dissertation was based on a straightforward finite element discretization of the spatial domain and a direct superposition of the matrix/fracture systems. Results from the discrete fracture model were compared with different homogenization methods where equivalent properties were generated and single-porosity simulations were performed.

For validation purposes, an explicit-fracture discretization was undertaken where a thin fracture was also divided into triangular finite elements. Results from this explicit representation, both in terms of saturation distributions and recovery were comparable to the discrete fracture representation of essentially the same domain. This comparison essentially validated the discrete fracture model.

No matter how rigorous, the homogenization methods failed to capture the saturation profiles observed in the discrete fracture case. The trends in recovery were matched by the mathematically vigorous homogenization, however homogenization method predicted much higher recoveries for all permeability contrasts. At higher resolution, homogenization predictions improved. If cross-permeability terms are included in simulations of homogenized data sets, the saturation maps are expected to better match the discrete fracture models.

At low permeability contrasts, the discrete fracture model showed more or less symmetric profiles for diagonally opposing injection-producer pair. At high permeability contrasts, the effect of fracture network was more pronounced. There was oil bypassing and lower recovery. This bypassing was enhanced at higher flow rates.

Parallel implementation of the discrete fracture model was extremely effective. Domain decomposition with elements overlapping between processors was the idea

employed. Superlinear performance was observed on both shared and distributed-shared memory machines. This was believed to be due to the memory cache problem.

APPENDIX

PARALLEL COMMUNICATION USING MPI

A.1 MPI library for the Parallel communication in Fortran programs

A.1.1 Basic communication functions in MPI

When the domain decomposition method applies to oil reservoir model, the data dependence in a processor to other processors requires the proper local and global data communications. For the grid cells, which are at the border of the processing nodes, data is required from the neighboring nodes. Generally, several data variables are packed into a temporary array to be sent to the neighboring nodes. The temporary array is then received and unpacked at the desired node. Separate subroutines were written to call the MPI library for the communication purposes. The regular send and receive message passing functions of MPI is explained in the following section.

The study used six basic functions offered in MPI that can be used to write any parallel program. In addition to these functions, MPI offers more functions to add flexibility, efficiency, robustness and convenience. The six basic functions are as follows.

- MPI_INIT
- MPI_COMM_SIZE
- MPI_COMM_RANK
- MPI_ISEND
- MPI_IRECV
- MPI_FINALIZE

Functions in MPI are specified by MPI and an underscore to let the compilers know that it belongs to the message passing library. The first step during the execution of a main program is initialization of MPI. MPI environment is established through

MPI_INIT. A header file which defines all the constants and variables supported by MPI is included during the initialization procedure. The initialization procedure lets the main program know that it has to link with the MPI library for the communication requirements during execution. Before communications can start, the communication configuration has to be defined. During message exchange between various processors, the messages are identified by tags. As the applications vary, so do the definitions and choices of tags. During the implementation of a parallel program, the user defines the number of processors to be used. The total number of processors specified by a user are identified through MPI_COMM_SIZE. This call determines the total number of processors associated with the communicator MPI_COMM_WORLD during execution of a particular program. The number of processors quantified through MPI_COMM_SIZE belongs to a group that is associated with that program. Within the group the processors are ranked from 0 to n-1, where n is the total number of processors specified by user. Each processor finds its own rank by calling MPI_COMM_RANK. Once the communication configuration is specified, the program can perform the message passing operations as required during execution. The two basic operations for message passing are sending data and receiving it. On the most parallel computers, moving data from one process to another takes more time than moving within a single process. To keep a program from being lowed down, many parallel computers allow users to start sending several messages and to proceed with other operations. MPI supports this approach by providing non-blocking sends and receives. The syntax of the operation is as follows.

MPI_ISEND(*buffer, count, datatype, destination, tag, comm, request, ierr*) : (*buffer, count, datatype*) defines the message to be sent; *count* occurrences of items of the form *datatype* starting at *buffer*. The *destination* is the rank of the processor to which the message needs to be sent. The processor belongs to the group associated with the communicator *comm*. The message is identified by the *tag* associated with it. The receiving processor with following command receives the message sent.

MPI_IRECV(*buffer, count, datatype, source, tag, comm, request, ierr*) : The message identified by the *tag* and originating from the processor with rank *source* will be received by the processor. The other arguments can be identified as described in MPI_ISEND. MPI_WAIT(*request, status, ierr*) may be used to wait for the completion of such a send and receive operation. If the message is received successfully an error free *status* is returned. Once the program is successfully completed, the MPI environment is terminated by the call MPI_FINALIZE. Once this command is executed, no MPI calls can be executed. The above mentioned basic commands can be used to develop a parallel version of any program.

A.1.2 Cartesian domain allocation in MPI

The study used the routine of MPI for manipulating Cartesian domain decomposition in parallel computer. The description of the routine is as follows. MPI_CART_CREATE(*comm, ndim, dims, periods, reorder, comm2d, ierr*) The routine creates a Cartesian decomposition of the processes, with the number of dimensions given by the *ndim* argument. The number of processes in any direction can be specified by giving a positive value to the corresponding element of *dims.* The *periods* argument

indicates whether the processes at the ends are connected. By setting the argument *reorder* to *.true.*, MPI finds a good way to assign the process to the elements of the decomposition for better performance.

A.1.3 MPI derived datatype

For a regular grid system, message communication can be achieved more conveniently among multiple processors using the function of MPI derived datatype. Accompanied with the basic datatypes; MPI_INTEGER, MPI_REAL and MPI_DOUBLE_PRECISION, MPI allows a user-defined datatype specifying the length of message as given count of occurrences in noncontiguous areas in memory. It is a common situation in regular grid system to send or receive arrays separated in constant amount in memory. The description starts with defining MPI_TYPE_VECTOR as follows. MPI_TYPE_VECTOR(*count, blocklength, stride, oldtype, newtype, ierr*): The arguments to MPI_TYPE_VECTOR describe a block, which consists of a number of contiguous copies of the input datatype given by the second argument. The first argument is the number of blocks; the second is the number of elements of the old datatype in each block. The old datatype is the fourth argument. The third argument is the stride; this is the distance in terms of the extent of the input datatype between successive elements. The fifth argument is the created derived datatype. For an example, the following simple array shows how to set the new datatype.

16	17	18	19	20
11	12	13	14	15
6	7	8	9	10
1	2	3	4	5

MPI_TYPE_ VECTOR(3,1,5,MPI_REAL,nwtype,ierr)

To commit the new datatype to the system, the following routine must be called before it can be used. MPI_TYPE_COMMIT(*newtype, ierr*). When parallel program reaches to exit or a new datatype is not used more, the new datatype need to be free as follows. MPI_TYPE_FREE(*newtype, ierr*)

LITERATURE CITED

Amazinane, B. and A. Bourgeat, "Effective Behavior of Two-Phase Flow in Heterogeneous Reservoir," *IMA*, **11**, 1 (1988).

Aziz, K. and A. Settari, *Petroleum Reservoir Simulation*, Elsevier, London (1979).

Bear, J., "Modeling Flow and Contaminant Transport in Fractured Rocks," *Flow and Contaminant Transport in Fractured Rock*, J. Bear, Chin-Fu Tsang and G. de Marsily, eds., Academic Press, San Diego, p. 1 (1993).

Bird, R. B., W. E. Steward and E. N. Lightfoot, *Transport Phenomena*, John Wiley &Sons Inc. (1960).

Bourgeat, A., "Homogenization Method Applied to the Behavior of Naturally Fissured Reservoir," *Mathematical Methods in Energy Research*, K. I. Gross, ed., SIAM, p.181 (1984).

Chang, M. M., P. Sarathi, R. J. Heemstra, A. M. Cheng and J. F. Pautz, *User's Guide and Documentation Manual for BOAST-VHS for the PC*, U.S. Dept. of Energy Report NIPER-542 Distribution Category UC-122 (1992).

Dalen, V., "Simplified Finite Element Models for Reservoir Flow Problems," *SPE J.*, 333 (1979).

Dershowitz, W., G. Lee, J. Geier, T. Foxford, P. LaPointe and A. Thomas, *FRACMAN User Document: Interactive Discrete Fracture Data Analysis, Geometric Modeling, and Exploration Simulation*, Golder Associates, Inc. (1995).

Dowd, K., *High Performance Computing*, O'Reilly & Associates Inc., (1993).

Eisenstat, S. C. and H. F. Walker, "Choosing the Forcing Terms in an Inexact Newton Method," *SIAM J. Sci. Comput.*, **17**(1), 16 (1996).

Fletcher, R., "Conjugate Gradient Methods for Indefinite Systems," *Volume 506 of lecture notes in Mathematics*, Spring-Verlag, Heidelberg, p. 73 (1976).

Forster, C. B., D. L. Nielson and M. Deo, *Chractreization and Simulation of an Exhumed Fractured Petroleum Reservoir*, Final report, Subcontract number: GS4S1734 submitted to BDM-OKLAHOMA, Inc. (1998).

Fortune, S. J., "A Sweepline Algorithm for Voronoi Diagrams," *Algorithmica*, **2**, 153 (1987).

Freund, R. W., "A Ttranspose Free Quasi Minimal Residual Method for Non-Hermitian Linear System," *SIAM J. Sci. Comput*., **14**, 470 (1993).

Gropp, W., E. Lusk and A. Skjellum, *Using MPI : Portable Parallel Programming with the Message Passing Interface*, the MIT press, Cambridge, Massachusetts (1995).

Hestenes, M. R. and E. L. Stiefel, "Method of Conjugate Gradients for Solving Linear Systems," *Journal of research national bureau of standards*, **49**, 435 (1952).

Hofhaus, J. and E. F. Van De Velde, "Alternating Direction Line Relaxation Methods on Multicomputers," *SIAM J. Sci. Comput*., **17**(2), 454 (1996).

Huyakorn, P. A., S. Panday and Y. S. Wu, "A Three-Dimensional Multiphase Flow Model for Assessing NAPL Contamination in Porous and Fractued Media, 1. Formulation" *Journal of Contaminant Hydrology*, **16**, 109 (1994).

Huyakorn, P. S. and G. F. Pinder, *Computational Methods in Subsurface Flow*, Academic Press, Inc., London (1983).

Kaluarachchi, J. J. and J. C. Parker, "An Efficient Finite Element Method for Modeling Multiphase Flow," *Water Resources Research*, 25, 1, p. 43 (1989).

Kazemi, H., "Pressure Transient Analysis of Naturally Fractured Reservoirs with Uniform Fracture Distribution," *SPE. J.*, 451, (1969).

Kazemi, H. and J. R. Gilman, "Multiphase Flow in Fractured Petroleum Reservoirs," *Flow and Contaminant Transport in Fractured Rock*, J. Bear, Chin-Fu Tsang and G. de Marsily, eds., Academic Press, San Diego, p. 267 (1993).

Kelley, C. T., *Iterative Methods for Linear and Nonlinear Equations*, SIAM, Philadelphia, p. 33 (1995).

Keunings, R., "Parallel Finite Element Algorithms Applied to Computational Reology," *Computers Chem. Eng.*, **19**(6), 647 (1995).

Killough, J. E., "Is Parallel Computing Ready for Reservoir Simulation ? a Critical Analysis of the State of the Art," *paper SPE 26634*, Society of Petroleum Engineers, Dallas, TX (1993).

Killough, J. E., "The Application of Parallel Computing to The Flow of Fluids in Porous Media," *Computers Chem. Eng.*, **19**(6), 775 (1995).

Killough, J. E. and B. Rao, "Simulation of Compositional Reservoir Phenomena on a Distributed-Memory Parallel Computer," JPT, 1368 (1991).

Koebbe, J., *HomCode. a Code for Scaling Up Permeabilities Using Homogenization, internal communication* (1998).

Mattax, C. C. and J. R. Kyte, "Simplified Finite Element Models for Reservoir Flow Problems," *paper SPE 187*, Society of Petroleum Engineers, Dallas, TX (1961).

Pinder G. F., P. S. Huyarkorn and E. A. Sudicky, "Simulation of Flow and Transport in Fractured Porous Media," *Flow and Contaminant Transport in Fractured Rock*, J. Bear, Chin-Fu Tsang and G. de Marsily, eds., Academic Press, San Diego, p. 396 (1993).

Press, W. H., S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, *Numerical Recipes in Fortran*, Cambridge University Press, p. 383 (1992).

Rame, M. and M. Delshad, "A Compositional Reservoir Simulator on Distributed Memory Parallel Computers," *paper SPE 29103*, Society of Petroleum Engineers, Dallas, TX (1995).

Saad, Y. and M. H. Schultz, "GMRES: a Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems," *SIAM J. Sci. Comput*., **7**, 856 (1986).

Slough, K. J., F. A. Sudicky and P. A. Forsyth, "Importance of Rock Matrix Entry Pressure on DNAPL Migration in Fractured Geologic Materials," *Ground Water*, **37**, 2 (1999).

Terez, I. E. and Firoozabadi Abbas "Water Injection in Water-Wet Fractued Porous Media: Experiments and a New Model With Modified Buckley- Leverett Theory," *SPE J.*, **4**, 2 (1999).

Van Der Vorst, H. A., "BiCGSTAB. a Fast and Smoothly Converging Variant of Bi- CG for the Solution of Nonsymmetric Linear Systems," *SIAM J. Sci. Comput*., **13**, 631 (1992).

Watts, J. W., "Reservoir Simulation: Past, Present, and Future," *paper SPE 38441*, Society of Petroleum Engineers, Dallas, TX (1997).

Young, L. C. and K. Hemanth-Kumar, "High-Performance Black Oil Computations," *paper SPE 21215*, Society of Petroleum Engineers, Dallas, TX (1991).